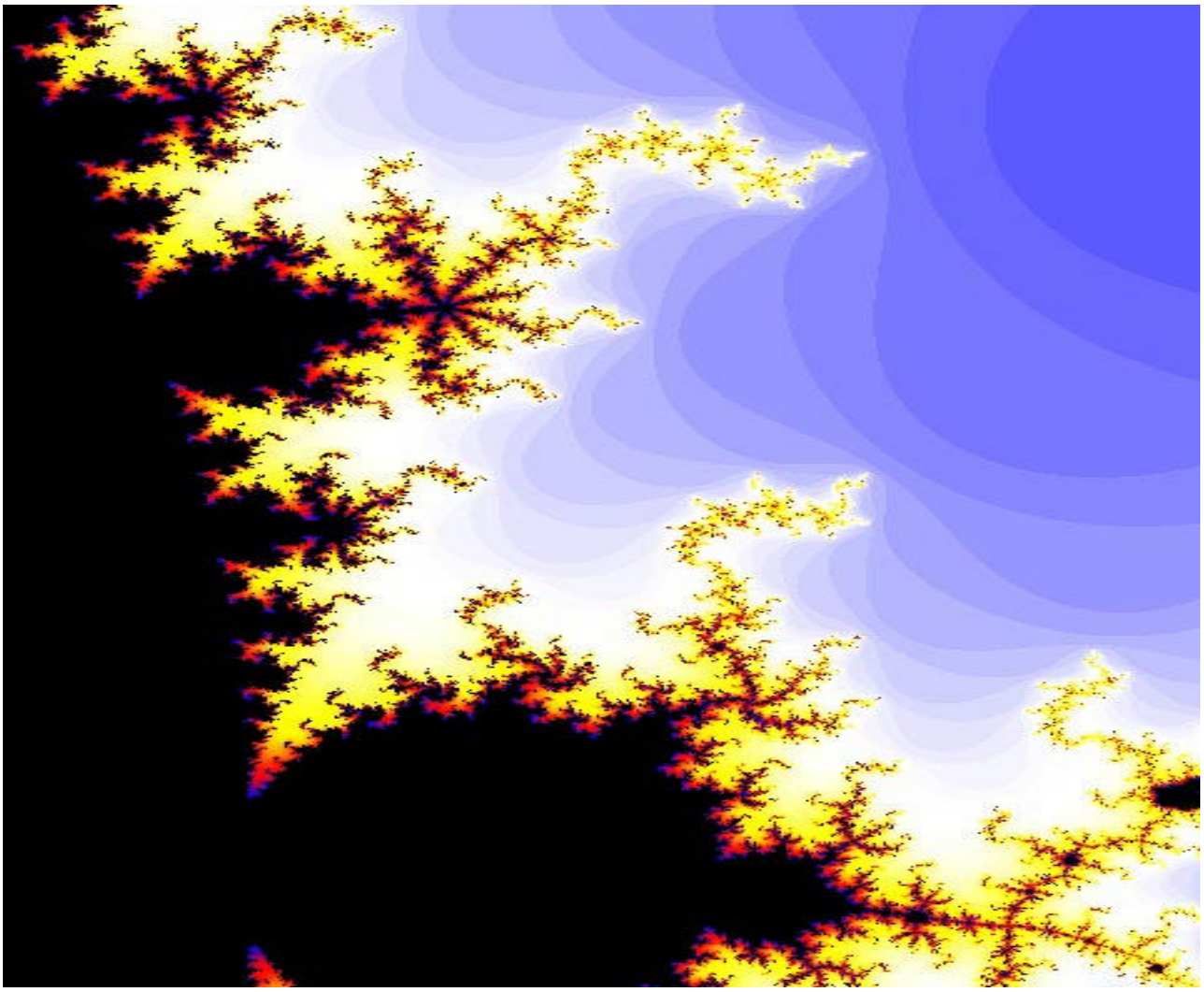
VISIT…

VISIT…

An Introduction to Pattern Recognition

Michael Alder

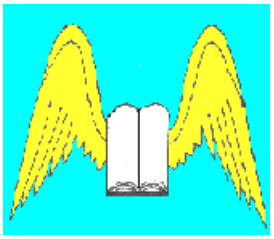


HeavenForBooks.com

An Introduction to Pattern Recognition

by

Michael Alder



HeavenForBooks.com

An Introduction to Pattern Recognition

This Edition ©Mike Alder, 2001

Warning: This edition is not to be copied, transmitted excerpted or printed except on terms authorised by the publisher

HeavenForBooks.com

Next Up Previous Contents

Next: Contents

An Introduction to Pattern Recognition: Statistical, Neural Net and Syntactic methods of getting robots to see and hear.

Michael D. Alder

September 19, 1997

Preface

Automation, the use of robots in industry, has not progressed with the speed that many had hoped it would. The forecasts of twenty years ago are looking fairly silly today: the fact that they were produced largely by journalists for the benefit of boardrooms of accountants and MBA's may have something to do with this, but the question of why so little has been accomplished remains.

The problems were, of course, harder than they looked to naive optimists. Robots have been built that can move around on wheels or legs, robots of a sort are used on production lines for routine tasks such as welding. But a robot that can clear the table, throw the eggshells in with the garbage and wash up the dishes, instead of washing up the eggshells and throwing the dishes in the garbage, is still some distance off.

Pattern Classification, more often called *Pattern Recognition*, is the primary bottleneck in the task of automation. Robots without sensors have their uses, but they are limited and dangerous. In fact one might plausibly argue that a robot without sensors isn't a *real* robot at all, whatever the hardware manufacturers may say. But equipping a robot with vision is easy only at the hardware level. It is neither expensive nor technically difficult to connect a camera and frame grabber board to a computer, the robot's `brain'. The problem is with the software, or more exactly with the *algorithms* which have to decide what the robot is looking at; the input is an array of pixels, coloured dots, the software has to decide whether this is an image of an eggshell or a teacup. A task which human beings can master by age eight, when they decode the firing of the different light receptors in the retina of the eye, this is computationally very difficult, and we have only the crudest ideas of how it is done. At the hardware level there are marked similarities between the eye and a camera (although there are differences too). At the algorithmic level, we have only a shallow understanding of the issues.

Human beings are very good at learning a large amount of information about the universe and how it can be treated; transferring this information to a program tends to be slow if not impossible.

This has been apparent for some time, and a great deal of effort has been put into research into practical methods of getting robots to recognise things in images and sounds. The Centre for Intelligent Information Processing Systems (CIIPS), of the University of Western Australia, has been working in the area for some years now. We have been particularly concerned with neural nets and applications to pattern recognition in speech and vision, because adaptive or *learning* methods are clearly of great potential value. The present book has been used as a postgraduate textbook at CIIPS for a Master's level course in Pattern Recognition. The contents of the book are therefore oriented largely to image and to some extent speech pattern recognition, with some concentration on neural net methods.

Students who did the course for which this book was originally written, also completed units in Automatic Speech Recognition Algorithms, Engineering Mathematics (covering elements of Information Theory, Coding Theory and Linear and Multilinear algebra), Artificial Neural Nets, Image Processing, Sensors and Instrumentation and Adaptive Filtering. There is some overlap in the material of this book and several of the other courses, but it has been kept to a minimum. Examination for the Pattern Recognition course consisted of a sequence of four micro-projects which together made up one mini-project.

Since the students for whom this book was written had a variety of backgrounds, it is intended to be accessible. Since the major obstructions to further progress seem to be fundamental, it seems pointless to try to produce a handbook of methods without analysis. Engineering works well when it is founded on some well understood scientific basis, and it turns into alchemy and witchcraft when this is not the case. The situation at present in respect of our scientific basis is that it is, like the curate's egg, good in parts. We are solidly grounded at the hardware level. On the other hand, the software tools for encoding algorithms (C, C++, MatLab) are fairly primitive, and our grasp of what algorithms to use is negligible. I have tried therefore to focus on the ideas and the (limited) extent to which they work, since progress is likely to require new ideas, which in turn requires us to have a fair grasp of what the old ideas are. The belief that engineers as a class are not intelligent enough to grasp any ideas at all, and must be trained to jump through hoops, although common among mathematicians, is not one which attracts my sympathy.

Instead of exposing the fundamental ideas in algebra (which in these degenerate days is less intelligible than Latin) I therefore try to make them plain in English.

There is a risk in this; the ideas of science or engineering are quite diferent from those of philosophy (as practised in these degenerate days) or literary criticism (ditto). I don't mean they are about different things, they are different in kind. Newton wrote `Hypotheses non fingo', which literally translates as `I do not make hypotheses', which is of course quite untrue, he made up some spectacularly successful hypotheses, such as universal gravitation. The difference between the two statements is partly in the hypotheses and partly in the fingo. Newton's `hypotheses' could be tested by observation or calculation, whereas the explanations of, say, optics, given in Lucretius *De Rerum Naturae* were recognisably `philosophical' in the sense that they resembled the writings of many contemporary philosophers and literary critics. They may persuade, they may give the sensation of profound insight, but they do not reduce to some essentially prosaic routine for determining if they are actually true, or at least useful. Newton's did. This was one of the great philosophical advances made by Newton, and it has been underestimated by philosophers since.

The reader should therefore approach the discussion about the underlying ideas with the attitude of irreverence and disrespect that most engineers, quite properly, bring to non-technical prose. He should ask: what procedures does this lead to, and how may they be tested? We deal with high level abstractions, but they are aimed always at reducing our understanding of something prodigiously complicated to something simple.

It is necessary to make some assumptions about the reader and only fair to say what they are.

I assume, first, that the reader has a tolerably good grasp of Linear Algebra concepts. The concepts are more important than the techniques of matrix manipulation, because there are excellent packages which can do the calculations if you know what to compute. There is a splendid book on Linear Algebra available from the publisher HeavenForBooks.com

I assume, second, a moderate familiarity with elementary ideas of Statistics, and also of contemporary Mathematical notation such as any Engineer or Scientist will have encountered in a modern undergraduate course. I found it necessary in this book to deal with underlying ideas of Statistics which are seldom mentioned in undergraduate courses.

I assume, finally, the kind of general exposure to computing terminology familiar to anyone who can read, say, *Byte* magazine, and also that the reader can program in C or some similar language.

I do not assume the reader is of the male sex. I usually use the pronoun `he' in referring to the reader because it saves a letter and is the convention for the generic case. The proposition that this will depress some women readers to the point where they will give up reading and go off and become subservient housewives does not strike me as sufficiently plausible to be worth considering further.

This is intended to be a happy, friendly book. It is written in an informal, one might almost say breezy, manner, which might irritate the humourless and those possessed of a conviction that intellectual respectability entails stuffiness. I used to believe that all academic books on difficult subjects were obliged for some mysterious reason to be oppressive, but a survey of the better writers of the past has shown me that this is in fact a contemporary habit and in my view a bad one. I have therefore chosen to abandon a convention which must drive intelligent people away from Science and Engineering in large numbers.

The book has jokes, opinionated remarks and pungent value judgments in it, which might serve to entertain readers and keep them on their toes, so to speak. They may also irritate a few who believe that the pretence that the writer has no opinions should be maintained even at the cost of making the book boring. What this convention usually accomplishes is a sort of bland porridge which discourages critical thought about fundamental assumptions, and thought about fundamental assumptions is precisely what this area badly needs.

So I make no apology for the occasional provocative judgement; argue with me if you disagree. It is quite easy to do that via the net, and since I enjoy arguing (it is a pleasant game), most of my provocations are deliberate. Disagreeing with people in an amiable, friendly way, and learning something about why people feel the way they do, is an important part of an education; merely learning the correct things to say doesn't get you very far in Mathematics, Science or Engineering. Cultured men or women should be able to dissent with poise, to refute the argument without losing the friend.

The judgements are, of course, my own; CIIPS and the Mathematics Department and I are not responsible for each other. Nor is it to be expected that the University of Western Australia should ensure that my views are politically correct. If it did that, it wouldn't be a university. In a good university, It is a case of *Tot homines, quot sententiae*, there are as many opinions as people. Sometimes more!

I am most grateful to my colleagues and students at the Centre for assistance in many forms; I have shamelessly borrowed their work as examples of the principles discussed herein. I must mention Dr. Chris deSilva with whom I have worked over many years, Dr. Gek Lim whose energy and enthusiasm for Quadratic Neural Nets has enabled them to become demonstrably useful, and Professor Yianni Attikiouzel, director of CIIPS, without whom neither this book nor the course would have come into existence.

- Contents
- Basic Concepts
  - Measurement and Representation
    - From objects to points in space
    - Telling the guys from the gals
    - Paradigms
  - Decisions, decisions..
    - Metric Methods
    - Neural Net Methods (Old Style)
    - Statistical Methods
      - Parametric
      - Non-parametric
    - CART et al
  - Clustering: supervised v unsupervised learning
  - Dynamic Patterns
  - Structured Patterns
  - Alternative Representations

        - Strings, propositions, predicates and logic
        - Fuzzy Thinking
        - Robots
    - Summary of this chapter
    - Exercises
    - Bibliography
- Image Measurements
    - Preliminaries
        - Image File Formats
    - Generalities
    - Image segmentation: finding the objects
        - Mathematical Morphology
        - Little Boxes
        - Border Tracing
        - Conclusions on Segmentation
    - Measurement Principles
        - Issues and methods
        - Invariance in practice
    - Measurement practice
        - Quick and Dumb
        - Scanline intersections and weights
        - Moments
        - Zernike moments and the FFT
            - Historical Note
        - Masks and templates
        - Invariants
        - Simplifications and Complications
    - Syntactic Methods
    - Summary of OCR Measurement Methods
    - Other Kinds of Binary Image
    - Greyscale images of characters
        - Segmentation: Edge Detection
    - Greyscale Images in general

        - Segmentation
        - Measuring Greyscale Images
        - Quantisation
        - Textures
    - Colour Images
        - Generalities
        - Quantisation
        - Edge detection
        - Markov Random Fields
        - Measurements
    - Spot counting
    - IR and acoustic Images
    - Quasi-Images
    - Dynamic Images
    - Summary of Chapter Two
    - Exercises
    - Bibliography
- Statistical Ideas
    - History, and Deep Philosophical Stuff
        - The Origins of Probability: random variables
        - Histograms and Probability Density Functions
        - Models and Probabilistic Models
    - Probabilistic Models as Data Compression Schemes
        - Models and Data: Some models are better than others
    - Maximum Likelihood Models
        - Where do Models come from?
    - Bayesian Methods
        - Bayes' Theorem
        - Bayesian Statistics
        - Subjective Bayesians
    - Minimum Description Length Models
        - Codes: Information theoretic preliminaries
        - Compression for coin models

        - Compression for *pdf*s
        - Summary of Rissanen Complexity
    - Summary of the chapter
    - Exercises
    - Bibliography
- Decisions: Statistical methods
    - The view into
    - Computing *PDF*s: Gaussians
        - One Gaussian per cluster
            - Dimension 2
        - Lots of Gaussians: The EM algorithm
            - The EM algorithm for Gaussian Mixture Modelling
        - Other Possibilities
    - Bayesian Decision
        - Cost Functions
        - Non-parametric Bayes Decisions
        - Other Metrics
    - How many things in the mix?
        - Overhead
        - Example
        - The Akaike Information Criterion
        - Problems with EM
    - Summary of Chapter
    - Exercises
    - Bibliography
- Decisions: Neural Nets(Old Style)
    - History: the good old days
        - The Dawn of Neural Nets
        - The death of Neural Nets
        - The Rebirth of Neural Nets
        - The End of History
    - Training the Perceptron
        - The Perceptron Training Rule

- Committees
  - Committees and XOR
  - Training Committees
  - Capacities of Committees: generalised XOR
  - Four Layer Nets
  - Building up functions
- Smooth thresholding functions
  - Back-Propagation
  - Mysteries of Functional Analysis
  - Committees vs Back-Propagation
- Compression: is the model worth the computation?
- Other types of (Classical) net
  - General Issues
  - The Kohonen Net
  - Probabilistic Neural Nets
  - Hopfield Networks
    - Introduction
    - Network Characteristics
    - Network Operation
    - The Network Equations
    - Theory of the Network
    - Applications
  - The Boltzmann Machine
    - Introduction
    - Simulated Annealing
    - Network Characteristics
    - Network Operation
    - Theory of the Network
    - Applications
  - Bidirectional Associative Memory
    - Introduction
    - Network Characteristics
    - Network Operation

- The Network Equations
- Theory of the Network
- Applications
- ART
  - Introduction
  - Network Characteristics
  - Network Operation
  - Theory of the Network
  - Applications
- Neocognitron
  - Introduction
  - Network Structure
  - The Network Equations
  - Training the Network
  - Applications
- References
- Quadratic Neural Nets: issues

- Summary of Chapter Five
- Exercises
- Bibliography

- Continuous Dynamic Patterns
  - Automatic Speech Recognition
    - Talking into a microphone
    - Traditional methods: VQ and HMM
      - The Baum-Welch and Viterbi Algorithms for Hidden Markov Models
    - Network Topology and Initialisation
    - Invariance
    - Other HMM applications
    - Connected and Continuous Speech
  - Filters
    - Linear Systems
    - Moving Average Filters
    - Autoregressive Time Series

        - Linear Predictive Coding or ARMA modelling
        - Into $\mathbb{R}^n$
        - States
        - Wiener Filters
        - Adaptive Filters, Kalman Filters
    - Fundamentals of dynamic patterns
    - Exercises
    - Bibliography
- Discrete Dynamic Patterns
    - Alphabets, Languages and Grammars
        - Definitions and Examples
        - ReWrite Grammars
        - Grammatical Inference
        - Inference of ReWrite grammars
    - Streams, predictors and smoothers
    - Chunking by Entropy
    - Stochastic Equivalence
    - Quasi-Linguistic Streams
    - Graphs and Diagram Grammars
    - Exercises
    - Bibliography
- Syntactic Pattern Recognition
    - Precursors
    - Linear Images
    - Curved Elements
    - Parameter Regimes
    - Invariance:
      Classifying Transformations
    - Intrinsic and Extrisic Chunking (Binding)
    - Backtrack
    - Occlusion and other metric matters
    - Neural Modelling
        - Self-Tuning Neurons

        - Geometry and Dynamics
        - Extensions to Higher Order Statistics
        - Layering
    - Summary of Chapter
    - Exercises
    - Bibliography
- About this document ...

Next: Basic Concepts **Up:** An Introduction to Pattern **Previous:** An Introduction to Pattern

Contents

- Contents
- Basic Concepts
  - Measurement and Representation
    - From objects to points in space
    - Telling the guys from the gals
    - Paradigms
  - Decisions, decisions..
    - Metric Methods
    - Neural Net Methods (Old Style)
    - Statistical Methods
      - Parametric
      - Non-parametric
    - CART et al
  - Clustering: supervised v unsupervised learning
  - Dynamic Patterns
  - Structured Patterns
  - Alternative Representations
    - Strings, propositions, predicates and logic
    - Fuzzy Thinking
    - Robots
  - Summary of this chapter
  - Exercises
  - Bibliography
- Image Measurements
  - Preliminaries
    - Image File Formats
  - Generalities

- Image segmentation: finding the objects
  - Mathematical Morphology
  - Little Boxes
  - Border Tracing
  - Conclusions on Segmentation
- Measurement Principles
  - Issues and methods
  - Invariance in practice
- Measurement practice
  - Quick and Dumb
  - Scanline intersections and weights
  - Moments
  - Zernike moments and the FFT
    - Historical Note
  - Masks and templates
  - Invariants
  - Chaincoding
- Syntactic Methods
- Summary of OCR Measurement Methods
- Other Kinds of Binary Image
- Greyscale images of characters
  - Segmentation: Edge Detection
- Greyscale Images in general
  - Segmentation
  - Measuring Greyscale Images
  - Quantisation
  - Textures
- Colour Images
  - Generalities
  - Quantisation
  - Edge detection
  - Markov Random Fields
  - Measurements

- Spot counting
- IR and acoustic Images
- Quasi-Images
- Dynamic Images
- Summary of Chapter Two
- Exercises
- Bibliography

- Statistical Ideas
  - History, and Deep Philosophical Stuff
    - The Origins of Probability: random variables
    - Histograms and Probability Density Functions
    - Models and Probabilistic Models
  - Probabilistic Models as Data Compression Schemes
    - Models and Data: Some models are better than others
  - Maximum Likelihood Models
    - Where do Models come from?
  - Bayesian Methods
    - Bayes' Theorem
    - Bayesian Statistics
    - Subjective Bayesians
  - Minimum Description Length Models
    - Codes: Information theoretic preliminaries
    - Compression for coin models
    - Compression for *pdf*s
    - Summary of Rissanen Complexity
  - Summary of the chapter
  - Exercises
  - Bibliography
- Decisions: Statistical methods
  - The view into
  - Computing *PDF*s: Gaussians
    - One Gaussian per cluster
      - Dimension 2

        - Lots of Gaussians: The EM algorithm
            - The EM algorithm for Gaussian Mixture Modelling
        - Other Possibilities
    - Bayesian Decision
        - Cost Functions
        - Non-parametric Bayes Decisions
        - Other Metrics
    - How many things in the mix?
        - Overhead
        - Example
        - The Akaike Information Criterion
        - Problems with EM
    - Summary of Chapter
    - Exercises
    - Bibliography
- Decisions: Neural Nets(Old Style)
    - History: the good old days
        - The Dawn of Neural Nets
        - The death of Neural Nets
        - The Rebirth of Neural Nets
        - The End of History
    - Training the Perceptron
        - The Perceptron Training Rule
    - Committees
        - Committees and XOR
        - Training Committees
        - Capacities of Committees: generalised XOR
        - Four Layer Nets
        - Building up functions
    - Smooth thresholding functions
        - Back-Propagation
        - Mysteries of Functional Analysis
        - Committees vs Back-Propagation

- Compression: is the model worth the computation?
- Other types of (Classical) net
  - General Issues
  - The Kohonen Net
  - Probabilistic Neural Nets
  - Hopfield Networks
    - Introduction
    - Network Characteristics
    - Network Operation
    - The Network Equations
    - Theory of the Network
    - Applications
  - The Boltzmann Machine
    - Introduction
    - Simulated Annealing
    - Network Characteristics
    - Network Operation
    - Theory of the Network
    - Applications
  - Bidirectional Associative Memory
    - Introduction
    - Network Characteristics
    - Network Operation
    - The Network Equations
    - Theory of the Network
    - Applications
  - ART
    - Introduction
    - Network Characteristics
    - Network Operation
    - Theory of the Network
    - Applications
  - Neocognitron

- Introduction
- Network Structure
- The Network Equations
- Training the Network
- Applications
- References
- Quadratic Neural Nets: issues

- Summary of Chapter Five
- Exercises
- Bibliography

- Continuous Dynamic Patterns
  - Automatic Speech Recognition
    - Talking into a microphone
    - Traditional methods: VQ and HMM
      - The Baum-Welch and Viterbi Algorithms for Hidden Markov Models
    - Network Topology and Initialisation
    - Invariance
    - Other HMM applications
    - Connected and Continuous Speech
  - Filters
    - Linear Systems
    - Moving Average Filters
    - Autoregressive Time Series
    - Linear Predictive Coding or ARMA modelling
    - Into $\mathbb{R}^n$
    - States
    - Wiener Filters
    - Adaptive Filters, Kalman Filters
  - Fundamentals of dynamic patterns
  - Exercises
  - Bibliography
- Discrete Dynamic Patterns
  - Alphabets, Languages and Grammars

        - Definitions and Examples
        - ReWrite Grammars
        - Grammatical Inference
        - Inference of ReWrite grammars
    - Streams, predictors and smoothers
    - Chunking by Entropy
    - Stochastic Equivalence
    - Quasi-Linguistic Streams
    - Graphs and Diagram Grammars
    - Exercises
    - Bibliography
- Syntactic Pattern Recognition
    - Precursors
    - Linear Images
    - Curved Elements
    - Parameter Regimes
    - Invariance:
      Classifying Transformations
    - Intrinsic and Extrisic Chunking (Binding)
    - Backtrack
    - Occlusion and other metric matters
    - Neural Modelling
        - Self-Tuning Neurons
        - Geometry and Dynamics
        - Extensions to Higher Order Statistics
        - Layering
    - Summary of Chapter
    - Exercises
    - Bibliography

Mike Alder
9/19/1997

Next: Measurement and Representation **Up:** An Introduction to Pattern **Previous:** Contents

Basic Concepts

In this chapter I survey the scene in a leisurely and informal way, outlining ideas and avoiding the computational and the nitty gritty until such time as they can fall into place. We are concerned in chapter one with the overview from a great height, the synoptic perspective, the strategic issues. In other words, this is going to be a superficial introduction; it will be sketchy, chatty and may drive the reader who is expecting detail into frenzies of frustration. So put yourself in philosophical mode, undo your collar, loosen your tie, take off your shoes and put your feet up. Pour yourself a drink and get ready to think in airy generalities. The details come later.

- Measurement and Representation
  - From objects to points in space
  - Telling the guys from the gals
  - Paradigms
- Decisions, decisions..
  - Metric Methods
  - Neural Net Methods (Old Style)
  - Statistical Methods
    - Parametric
    - Non-parametric
  - CART et al
- Clustering: supervised v unsupervised learning
- Dynamic Patterns
- Structured Patterns
- Alternative Representations
  - Strings, propositions, predicates and logic
  - Fuzzy Thinking
  - Robots
- Summary of this chapter
- Exercises

- Bibliography

Mike Alder
9/19/1997

Next Up Previous Contents

Next: From objects to points **Up:** Basic Concepts **Previous:** Basic Concepts

Measurement and Representation

- From objects to points in space
- Telling the guys from the gals
- Paradigms

Mike Alder
9/19/1997

From objects to points in space

If you point a video camera at the world, you get back an array of *pixels* each with a particular gray level or colour. You might get a square array of 512 by 512 such pixels, and each pixel value would, on a gray scale, perhaps, be represented by a number between 0 (black) and 255 (white). If the image is in colour, there will be three such numbers for each of the pixels, say the intensity of red, blue and green at the pixel location. The numbers may change from system to system and from country to country, but you can expect to find, in each case, that the image may be described by an array of `real' numbers, or in mathematical terminology, a vector in $\mathbb{R}^n$ for some positive integer *n*. The number *n*, the length of the vector, can therefore be of the order of a million. To describe the image of the screen on which I am writing this text, which has 1024 by 1280 pixels and a lot of possible colours, I would need 3,932,160 numbers. This is rather more than the ordinary television screen, but about what High Definition Television will require.

An image on my monitor can, therefore, be coded as a vector in $\mathbb{R}^{3,932,160}$. A sequence of images such as would occur in a sixty second commercial sequenced at 25 frames a second, is a trajectory in this space. I don't say this is the best way to think of things, in fact it is a truly awful way (for reasons we shall come to), but it's one way.

More generally, when a scientist or engineer wants to say something about a physical system, he is less inclined to launch into a *haiku* or sonnet than he is to clap a set of measuring instruments on it, whether it be an electrical circuit, a steam boiler, or the solar system.

This set of instruments will usually produce a collection of numbers. In other words, the physical system gets coded as a vector in $\mathbb{R}^n$ for some positive integer *n*. The nature of the coding is clearly important, but once it has been set up, it doesn't change. By contrast, the measurements often do; we refer to this as the system changing in time. In real life, real numbers do not actually occur: decimal strings come in some limited length, numbers are specified to some precision. Since this precision can change, it is inconvenient to bother about what it is in some particular case, and we talk rather sloppily of vectors of real numbers.

I have known people who have claimed that $\mathbb{R}^n$ is quite useful when *n* is 1, 2 or 3, but that larger values were invented by Mathematicians only for the purpose of terrorising honest engineers and physicists, and can safely be ignored. Follow this advice at your peril.

It is worth pointing out, perhaps, that the representation of the states of a physical system as points in

$\mathbb{R}^n$ has been one of the great success stories of the world. Natural language has been found to be inadequate for talking about complicated things. Without going into a philosophical discursion about why this particular language works so well, two points may be worth considering. The first is that it separates two aspects of making sense of the world, it separates out the `world' from the properties of the measuring apparatus, making it easier to think about these things separately. The second is that it allows the power of geometric thinking, incorporating metric or more generally topological ideas, something which is much harder inside the discrete languages. The claim that `God is a Geometer', based upon the success of geometry in Physics, may be no more than the assertion that geometrical languages are better at talking about the world than non-geometrical ones. The general failure of Artificial Intellligence paradigms to crack the hard problems of how human beings process information may be in part due to the limitations of the language employed (often LISP!)

In the case of a microphone monitoring sound levels, there are many ways of coding the signal. It can be simply a matter of a voltage changing in time, that is, $n = 1$. Or we can take a Fourier Transform and obtain a simulated filter bank, or we can put the signal through a set of hardware filters. In these cases n may be, typically, anywhere between 12 and 256.

The system may change in continuous or discrete time, although since we are going to get the vectors into a computer at some point, we may take it that the continuously changing vector `signal' is discretely sampled at some appropriate rate. What *appropriate* means depends on the system. Sometimes it means once a microsecond, other times it means once a month.

We describe such dynamical systems in two ways; frequently we need to describe the law of time development, which is done by writing down a formula for a vector field, or as it used to be called, a *system of ordinary differential equations.* Sometimes we have to specify only some particular history of change: this is done formally by specifying a map from $\mathbb{R}$ representing time to the space $\mathbb{R}^n$ of possible states. We can simply list the vectors corresponding to different times, or we may be able to find a formula for calculating the vector output by the map when some time value is used as input to the map. It is both entertaining and instructive to consider the map:

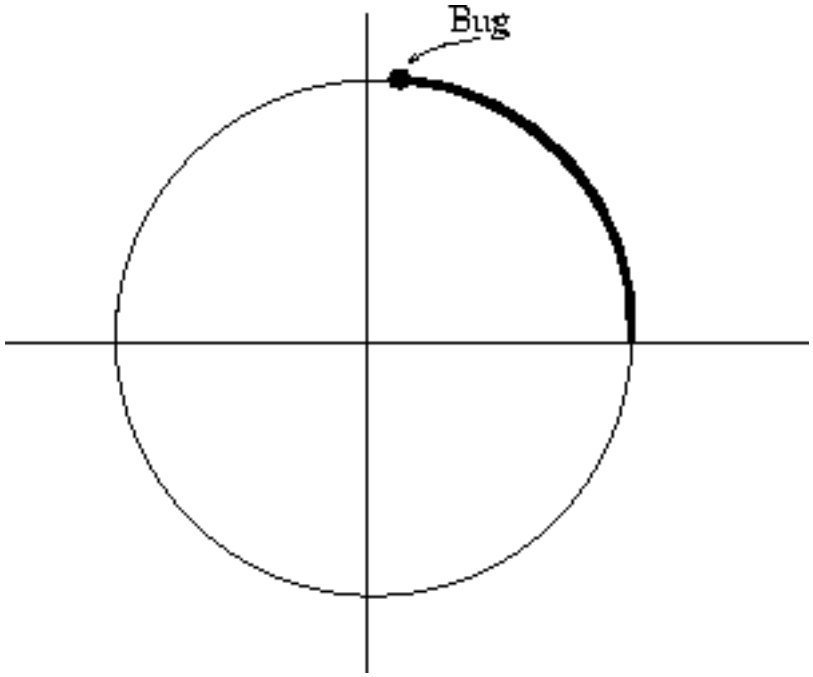
$$f : \mathbb{R} \longrightarrow \mathbb{R}^2$$

$$t \rightsquigarrow \begin{pmatrix} cos(t) \\ sin(t) \end{pmatrix}$$

If we imagine that at each time t between 0 and 2π a little bug is to be found at the location in $\mathbb{R}^2$ given by $f(t)$, then it is easy to see that the bug wanders around the unit circle at uniform speed, finishing up back where it started, at the location $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ after 2π time units. The terminology which we use to describe a bug moving in the two dimensional space $\mathbb{R}^2$ is the same as that used to describe a system

changing its state in the n-dimensional space $\mathbb{R}^n$. In particular, whether *n* is 2, 3 or a few million, we shall refer to a vector in $\mathbb{R}^n$ as a *point* in the space, and we shall make extensive use of the standard mathematician's trick of thinking of pictures in low dimensions while writing out the results of his thoughts in a form where the dimension is not even mentioned. This allows us to discuss an infinite number of problems at the same time, a very smart trick indeed. For those unused to it this is breathtaking, and the *hubris* involved makes beginners nervous, but one gets used to it.

Figure 1.1: A bug marching around the unit circle according to the map f.



This way of thinking is particularly useful when time is changing the state of the system we are trying to recognise, as would happen if one were trying to tell the difference between a bird and a butterfly by their motion in a video sequence, or more significantly if one is trying to distinguish between two spoken words. The two problems, telling birds from butterflies and telling a spoken `yes' from a `no', are very similar, but the representation space for the words is much higher than for the birds and butterflies. `Yes' and `no' are trajectories in a space of dimension, in our case, 12 or 16, whereas the bird and butterfly move in a three dimensional space and their motion is projected down to a two dimensional space by a video camera. We shall return to this when we come to discuss Automatic Speech Recognition.

Let us restrict attention for the time being, however, to the static case of a system where we are not much concerned with the time changing behaviour. Suppose we have some images of characters, say the letters

A

and

B

Then each of these, as pixel arrays, is a vector of dimension up to a million. If we wish to be able to say of a new image whether it is an A or a B, then our new image will also be a point in some rather high dimensional space. We have to decide which group it belongs with, the collection of points representing an A or the collection representing a B. There are better ways of representing such images as we shall see, but they will still involve points in vector spaces of dimension higher than 3.

So as to put our thoughts in order, we replace the problem of telling an image of an A from one of a B with a problem where it is much easier to visualise what is going on because the dimension is much lower. We consider the problem of telling men from women.

Next Up Previous Contents

Next: Telling the guys from **Up:** Measurement and Representation **Previous:** Measurement and Representation *Mike Alder*
9/19/1997

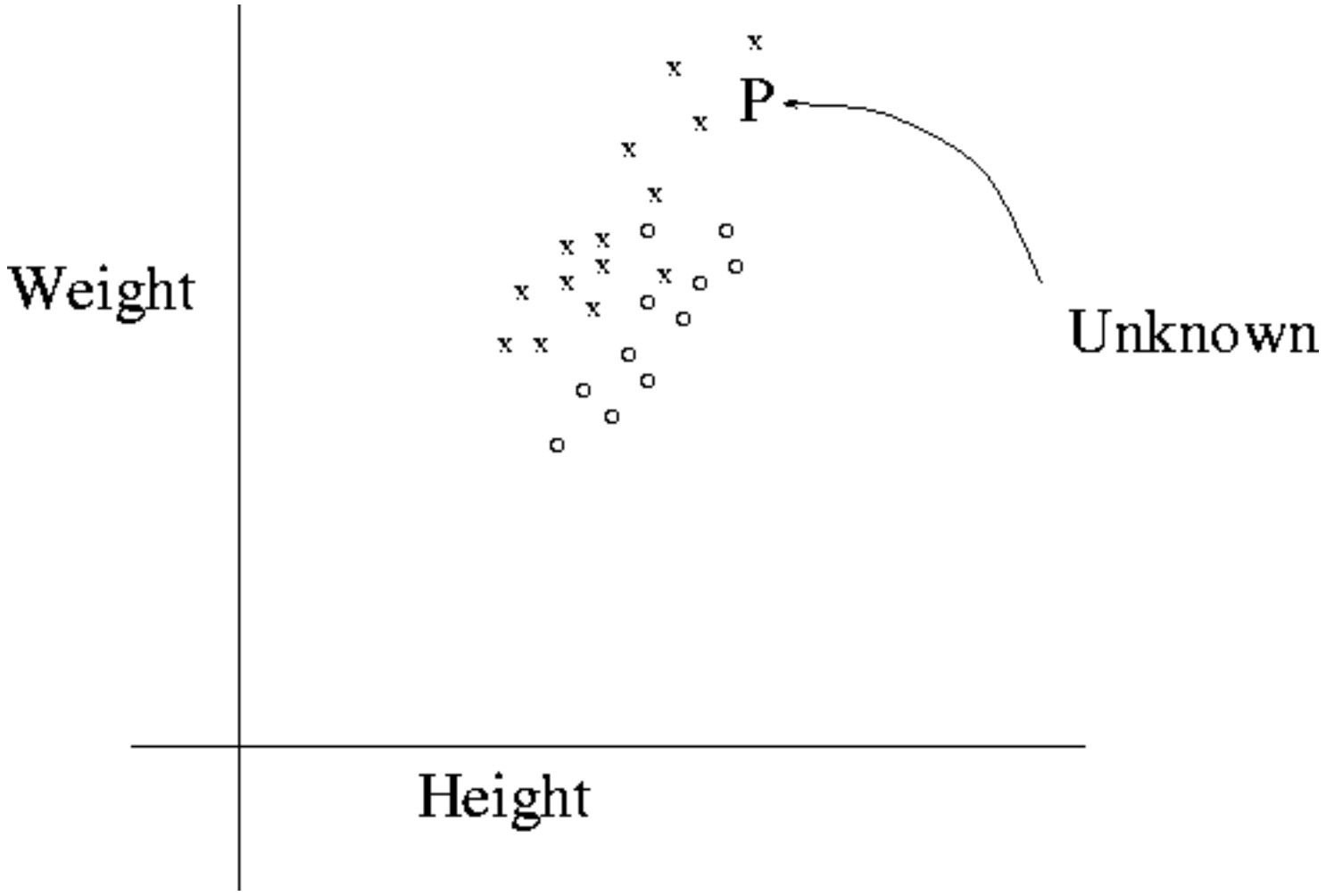
Next Up Previous Contents

Next: Paradigms **Up:** Measurement and Representation **Previous:** From objects to points

Telling the guys from the gals

Suppose we take a large number of men and measure their height and weight. We plot the results of our measurements by putting a point on a piece of paper for each man measured. I have marked a cross on *Fig.1.2.* for each man, in such a position that you can easily read off his weight and height. Well, you could do if I had been so thoughtful as to provide gradations and units. Now I take a large collection of women and perform the same measurements, and I plot the results by marking, for each woman, a circle.

Figure 1.2: X is male, O is female, what is P?



The results as indicated in *Fig.1.2.* are plausible in that they show that on average men are bigger than and heavier than women although there is a certain amount of overlap of the two samples. The diagram

also shows that tall people tend to be heavier than short people, which seems reasonable. Now suppose someone gives us the point P and assures us that it was obtained by making the usual measurements, in the same order, on some person not previously measured. The question is, do we think that the last person, marked by a P, is male or female?

There are, of course, better ways of telling, but they involve taking other measurements; it would be indelicate to specify what crosses my mind, and I leave it to the reader to devise something suitable. If this is all the data we have to go on, and we have to make a guess, what guess would be most sensible?

If instead of only two classes we had a larger number, also having, perhaps, horses and giraffes to distinguish, the problem would not be essentially different. If instead of working in dimension 2 as a result of choosing to measure only two attributes of the objects, men, women and maybe horses and giraffes, we were in dimension 12 as a result of choosing to measure twelve attributes, again the problem would be essentially the same- although it would be impracticable to draw a picture. I say it would be essentially the same; well it would be very different for a human being to make sense of lots of columns of numbers, but a computer program hasn't got eyes. The computer program has to be an embodiment of a set of rules which operates on a collection of columns of numbers, and the length of the column is not likely to be particularly vital. Any algorithm which will solve the two class, two dimensional case, should also solve the *k* class *n* dimensional case, with only minor modifications.

Next Up Previous Contents

Next: Paradigms **Up:** Measurement and Representation **Previous:** From objects to points *Mike Alder 9/19/1997*

Paradigms

The problem of telling the guys from the gals encapsulates a large part of Pattern Recognition. It may seem frivolous to put it in these terms, but the problem has all the essential content of the general problem (and it helps to focus the mind!) In general, we have a set of *objects* which human beings have decided belong into a finite number of *classes* or *categories*, for example, the objects might be human beings, or letters of the alphabet. We have some choice of *measuring process* which is applied to each object to turn it into a *point* in some space, or alternatively a *vector* or *array* of numbers. (If the vectors all have length *n* we say they are *n-dimensional*: 2 and 3 dimensional vectors correspond in an obvious way to points in a plane and in the space we live in by simply setting up a co-ordinate system. Hence the terminology.) So we have a set of *labelled points* in $\mathbb{R}^n$ for some *n*, where the label tells us what category the objects belong to. Now a new point is obtained by applying the measuring process to a new object, and the problem is to decide which class it should be assigned to.

There is a clear division of the problem of automatically recognising objects by machine into two parts. The first part is the measuring process. What are good things to measure? This is known in the jargon of the trade as the `feature selection problem', and the resulting $\mathbb{R}^n$ obtained is called the *feature space* for the problem.

A little thought suggests that this could be the hard part. One might reasonably conclude, after a little more thought, that there is no way a machine could be made which would be able to always measure the best possible things. Even if we restrict the problem to a machine which looks at the world, that is to dealing with images of things as the objects we want to recognise or classify, it seems impossible to say in advance what ought to be measured from the image in order to make the classification as reliable as possible. What is usually done is that a human being looks at some of the images, works out what he thinks the significant `features' are, and then tries to figure out a way of extracting numbers from images so as to capture quantitatively the amount of each `feature', thus mapping objects to points in the feature space, $\mathbb{R}^n$ for some *n*. This is obviously cheating, since ideally the machine ought to work out for itself, from the data, what these `features' are, but there are, as yet, no better procedures.

The second part is, having made some measurements on the image (or other object) and turned it into a point in a vector space, how does one calculate the class of a new point? What we need is some rule or *algorithm* because the data will be stored in a computer. The algorithm must somehow be able to compare, by some arithmetic/logical process, the new vector with the vectors where the class is known, and come out with a plausible guess.

Exercise!

It is a good idea to make these issues as concrete as possible, so you should, at this point, get some real data so as to focus the mind. This needs a kitchen weighing scales and a ruler, and a kitchen.

Get some eggs and some potatoes, For each egg first weigh it, write down its weight, then measure its greatest diameter, and write that down underneath. Repeat for all the eggs. This gives the egg list. Half a dozen (six) eggs should be enough.

Now do the same with a similar number of potatoes. This will give a potato list.

Plot the eggs on a piece of graph paper, just as for the guys and the gals, marking each one in red, repeat for the potatoes marking each as a point in blue.

Now take three objects from the kitchen at random (in my case, when I did this, I chose a coffee cup, a spoon and a box of matches); take another egg and another potato, make the same measurements on the five objects, and mark them on your graph paper in black.

Now how easy is it to tell the new egg from the new potatoe by looking at the graph paper? Can you see that all the other three objects are neither eggs nor potatoes? If the pairs of numbers were to be fed into a computer for a decision as to whether a new object is an egg or a potato, (or neither), what rule would *you* give the computer program for deciding?

What things *should* you have measured in order to reliably tell eggs from potatoes? Eggs from coffee-cups?

There are other issues which will cross the mind of the reflective reader: how did the human beings decide the actual categories in the first place? Don't laugh, but just how *do* you tell a man from a woman? By looking at them? In that case, your retinal cells and your brain cells between them must contain the information. If you came to an opinion about the best category to assign P in the problem of *Fig.1.2.* just by looking at it, what unarticulated rule did you apply to reach that conclusion? Could one articulate a rule that would agree with your judgement for a large range of cases of location of the new point P? Given any such rule, how does one persuade oneself that it is a good rule?

It is believed by almost all zoologists that an animal is a machine made out of meat, a robot constructed from colloids, and that this machine implements rules for processing sensory data with its brain in order to survive. This usually entails being able to classify images of other animals: your telling a man from a woman by looking is just a special case. We have then, an existence proof that the classification problems in which we are interested do in fact have solutions; the trouble is the algorithms are embedded in what is known in the trade as `wetware' and are difficult to extract from the brain of the user. Users of brains have been known to object to the suggestion, and anyway, nobody knows what to look for.

It is believed by some philosophers that the zoologists are wrong, and that minds do not work by any algorithmic processes. Since fruit bats can distinguish insects from thrown lumps of mud, either fruit bats have minds that work by non-algorithmic processes just like philosophers, or there is some fundamental difference between you telling a man from a woman and a fruit bat telling mud from insects, or the philosophers are babbling again. If one adopts the philosopher's position, one puts this book away and finds another way to pass the time. Now the philosopher may be right or he may be wrong; if he is right and you give up reading now, he will have saved you some heartbreak trying to solve an unsolvable problem. On the other hand, if he is right and if you continue with the book you will have a lot of fun even if you don't get to understand how brains work. If the philosopher is wrong and you give up, you will certainly have lost out on the fun and may lose out on a solution. So we conclude, by inexorable logic, that it is a mistake to listen to such philosophers, something which most engineers take as

axiomatic anyway.

Wonderful stuff logic, even if it *was* invented by a philosopher.

It is currently intellectually respectable to muse about the issue of how brains accomplish these tasks, and it is even more intellectually respectable (because harder) to experiment with suggested methods on a computer. If we take the view that brains somehow accomplish pattern classification or something rather like it, then it is of interest to make informed conjectures about how they do it, and one test of our conjectures is to see how well our algorithms perform in comparison with animals. We do not investigate the comparison in this book, but we do try to produce algorithms which can be so tested, and our algorithms are motivated by theoretical considerations and speculations on how brains do the same task. So we are doing Cognitive Science on the side. Having persuaded ourselves that the goal is noble and worthy of our energies, let us return to our muttons and start on the job of getting closer to that goal.

The usual way, as was explained above, of tackling the first part, of choosing a measuring process, is to leave it to the experimenter to devise one in any way he can. If he has chosen a good measuring process, then the second part will be easy: if the height and weight of the individual were the best you can do, telling men from women is hard, but if you choose to measure some other things, the two sets of points, the X's and O's, can be well separated and a new point P is either close to the X's or close to the O's or it isn't a human being at all. So you can tell retrospectively if your choice of what to measure was good or bad, up to a point. It not infrequently happens that all known choices are bad, which presents us with interesting issues. I shall return to this aspect of Pattern Recognition later when I treat *Syntactic* or *Structured* Pattern Recognition.

The second part assumes that we are dealing with (labelled) point sets in $\mathbb{R}^n$ belonging to two or more types. Then we seek a rule which gives us, for any new point, a label. There are lots of such rules. We consider a few in the next section.

Remember that you are supposed to be relaxed and casual at this stage, doing some general thinking and turning matters over in your mind! Can you think, in the light of eggs, potatoes and coffee-cups, of some simple rules for yourself?

Next Up Previous Contents

Next: Decisions, decisions.. **Up:** Measurement and Representation **Previous:** Telling the guys from *Mike Alder*
9/19/1997

Decisions, decisions..

- Metric Methods
- Neural Net Methods (Old Style)
- Statistical Methods
  - Parametric
  - Non-parametric
- CART et al

Mike Alder
9/19/1997

Metric Methods

One of the simplest methods is to find the closest point of the labelled set of points to the new point P, and assign to the new point whatever category the closest point has. So if (for the data set of guys and gals) the nearest point to P is an X, then we conclude that P should be a man. If a rationale is needed, we could argue that the measurement process is intended to extract important properties of the objects, and if we come out with values for the readings which are close together, then the objects must be similar. And if they are similar in respect of the measurements we have made, they ought, in any reasonable universe, to be similar in respect of the category they belong to as well. Of course it isn't clear that the universe we actually live in is the least bit reasonable.

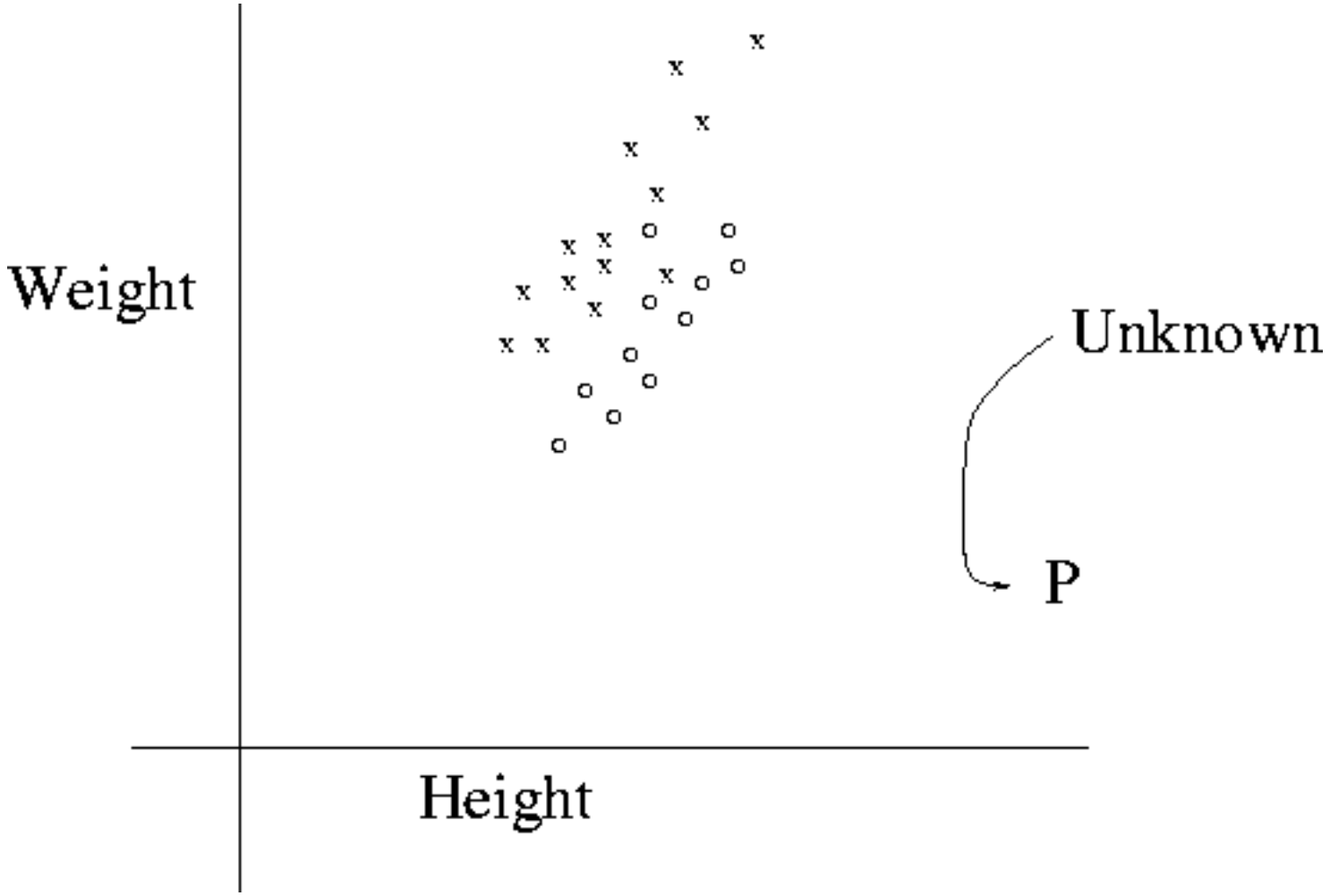
Such a rationale may help us devise the algorithm in the first place, but it may also allow us to persuade ourselves that the method is a good one. Such means of persuasion are unscientific and frowned upon in all the best circles. There are better ways of ensuring that it is a good method, namely testing to see how often it gives the right answer. It is noteworthy that no matter how appealing to the intuitions a method may be, there is an ultimate test which involves trying it out on real data. Of course, rationales tend to be very appealing to the intuitions of the person who thought of them, and less appealing to others. It is, however, worth reflecting on rationales, particularly after having looked at a bit more data; sometimes one can see the flaws in the rationales, and devise alternative methods.

The metric method is easy to implement in complete generality for *n* measurements, we just have to go through the whole list of points where we know the category and compute the distance from the given point P. How do we do this? Well, the usual Euclidean distance $d(\mathbf{x}, \mathbf{y})$ between the vectors

$$\begin{pmatrix} x^1 \\ x^2 \\ . \\ x^n \end{pmatrix} \text{ and } \begin{pmatrix} y^1 \\ y^2 \\ . \\ y^n \end{pmatrix}$$ is simply $\sqrt{\sum_{i=1}^{n}(x^i - y^i)^2}$, which is easy to compute. Now we

find that point **x** for which this distance from the new point P is a minimum. All that remains is to note its category. If anyone wants to know where the formula for the euclidean distance comes from in higher dimensions, it's a definition, and it gives the right answers in dimensions one, two and three. You have a better idea?

Figure 1.3: X is male, O is female, what is *this* P?

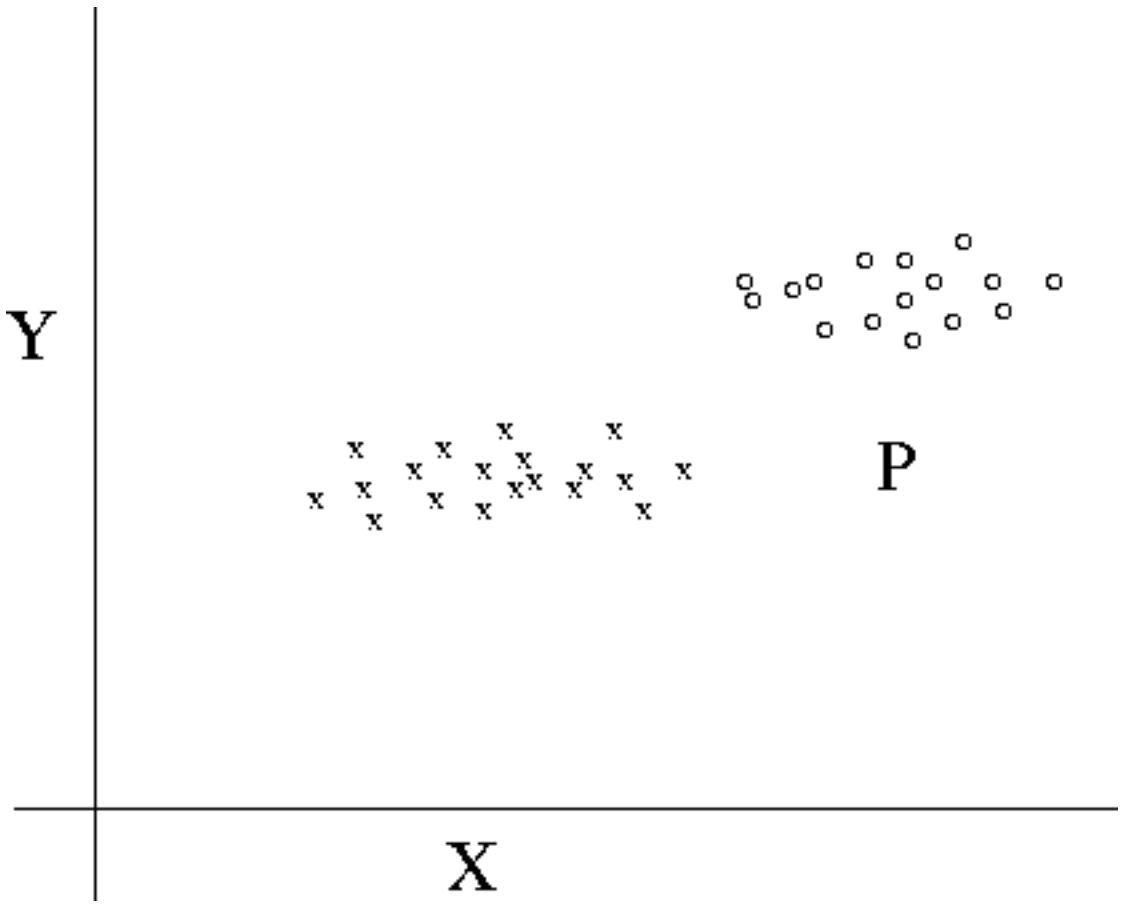


Reflection suggests some drawbacks. One is that we need to compute a comparison with all the data points in the set. This could be an awful lot. Another is, what do we do in a case such as *Fig.1.3.*, above, where the new point P doesn't look as if it belongs to either category? An algorithm which returns `Haven't the faintest idea, probably neither' when asked if the P of *Fig.1.3.* is a man or a woman would have some advantages, but the metric method needs some modification before it can do this. It is true that P is a long way from the closest point of either category, but how long is a long way?

Exercise: Is P in *Fig.1.3* likely to be (a) a kangaroo or (b) a pole vaulter's pole?

A more subtle objection would occur only to a geometer, a species of the genus Mathematician. It is this: why should you use the euclidean distance? What is so reasonable about taking the square root of the sum of the squares of the differences of the co-ordinates? Sure, it is what you are used to in two dimensions and three, but so what? If you had the data of *Fig.1.4.* for example, do you believe that the point P is, on the whole, `closer to' the X's or the O's?

Figure 1.4: Which is P closer to, the X's or the O's?



There is a case for saying that the X-axis in *Fig.1.4.* has been stretched out by something like three times the Y-axis, and so when measuring the distance, we should not give the X and Y coordinates the same weight. If we were to divide the X co-ordinates by 3, then P would be closer to the X's, whereas using the euclidean distance it is closer to the O's.

It can come as a nasty shock to the engineer to realise that there are an awful lot of different *metrics* (ways of measuring distances) on $\mathbb{R}^2$, and the old, easy one isn't necessarily the right one to use. But it should be obvious that if we measure weight in kilograms and height in centimetres, we shall get different answers from those we would obtain if we measured height in metres and weight in grams. Changing the measuring units in the above example changes the metric, a matter of very practical importance in real life. There are much more complicated cases than this which occur in practice, and we shall meet some in later sections, when we go over these ideas in detail.

Remember that this is only the mickey-mouse, simple and easy discussion on the core ideas and that the technicalities will come a little later.

Next Up Previous Contents

Next: Neural Net Methods (Old **Up:** Decisions, decisions.. **Previous:** Decisions, decisions.. *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Statistical Methods **Up:** Decisions, decisions.. **Previous:** Metric Methods

Neural Net Methods (Old Style)

Artificial Neural Nets have become very popular with engineers and computer scientists in recent times. Now that there are packages around which you can use without the faintest idea of what they are doing or how they are doing it, it is possible to be seduced by the name *neural nets*, into thinking that they must work in something like the way brains do. People who actually know the first thing about real brains and find out about the theory of the classical neural nets are a little incredulous that anyone should play with them. It is true that the connection with real neurons is tenuous in the extreme, and more attention should be given to the term *artificial*, but there are some connections with models of how brains work, and we shall return to this in a later chapter. Recall that in this chapter we are doing this once over briefly, so as to focus on the underlying ideas, and that at present we are concerned with working out how to think about the subject.

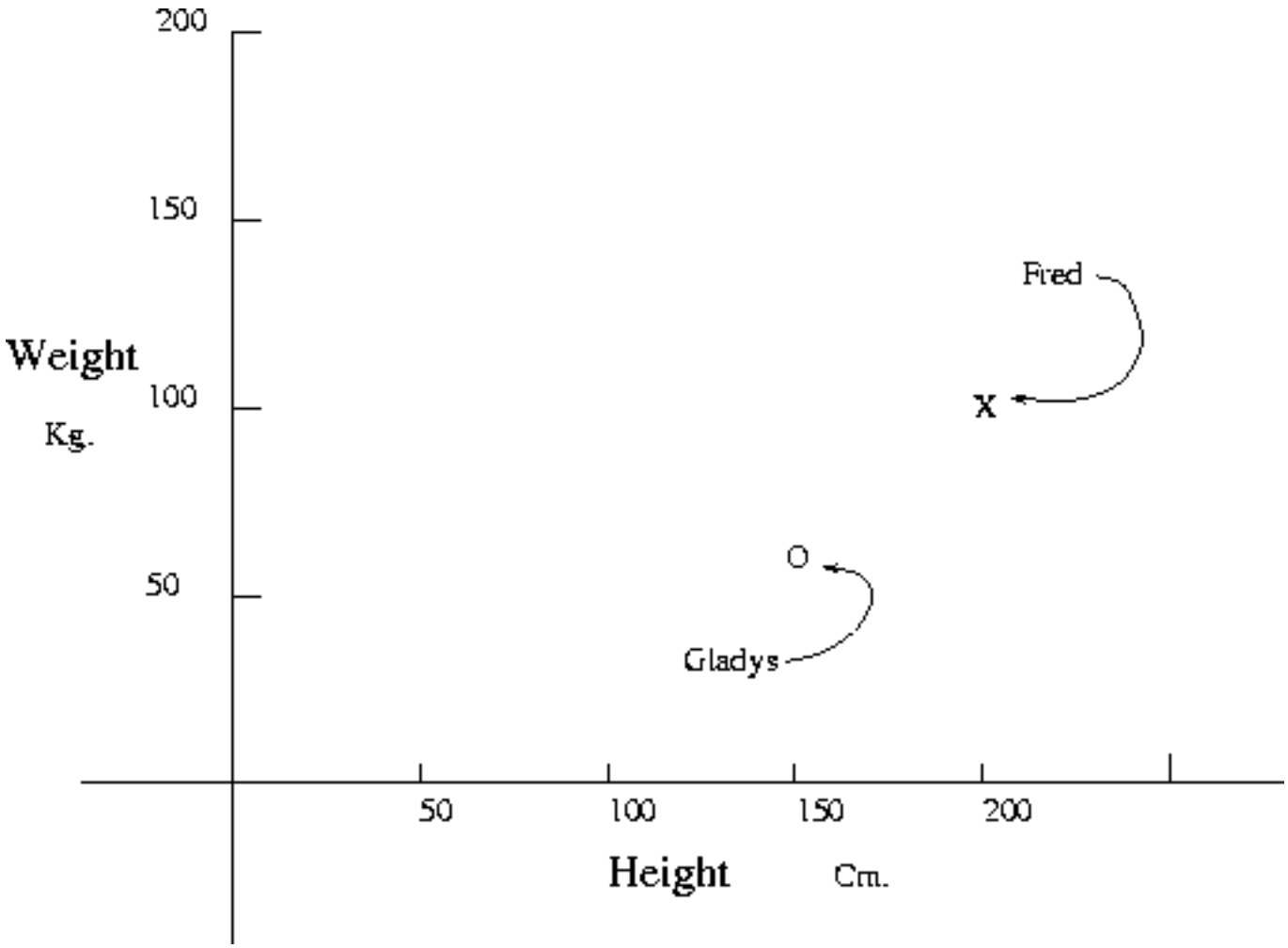
I shall discuss other forms of neural net later, here I focus on a particular type of net, the *Multilayer Perceptron* or *MLP*, in its simplest avatar.

We start with the single unit *perceptron* , otherwise a three layer neural net with one unit in the hidden layer. In order to keep the dimensions nice and low for the purposes of visualising what is going on, I shall recycle *Fig.1.2.* and use x and y for the height and weight values of a human being. I shall also assume that, initially, I have only two people in my data set, Fred who has a height of 200 cm and weighs in at 100 kg, and Gladys who has a height of 150 cm and a weight of 60 kg. We can picture them graphically as in *Fig.1.5.*, or algebraically as

$$Fred = \begin{pmatrix} 200 \\ 100 \end{pmatrix}$$

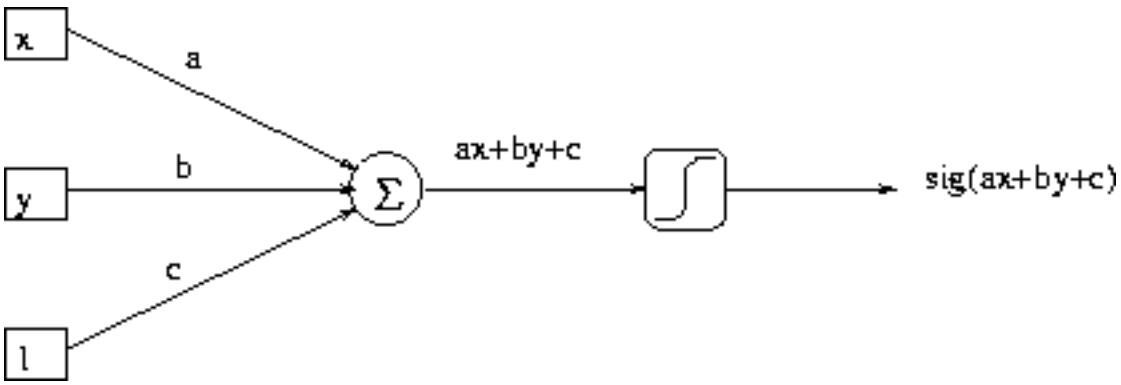
$$Gladys = \begin{pmatrix} 150 \\ 60 \end{pmatrix}$$

Figure 1.5: Gladys and Fred, abstracted to points in $\mathbb{R}^2$



The neural net we shall use to classify Fred and Gladys has a diagram as shown in *Fig.1.6.* The input to the net consists of two numbers, the height and weight, which we call *x* and *y*. There is a notional `fixed' input which is always 1, and which exists to represent a so called `threshold'. The square boxes represent the input to the net and are known in some of the Artificial Neural Net (ANN) literature as the first layer. The second layer in this example contains only one unit (believed in some quarters to represent a neuron) and is represented by a circle. The lines joining the first layer to the second layer have numbers attached. These are the *weights*, popularly supposed to represent the strength of synaptic connections to the neuron in the second layer from the input or sensory layer.

Figure 1.6: A very simple neural net in two dimensions



The node simply sums up the weighted inputs, and if the weights are a, b and c, as indicated, then the output is *ax+by+c* when the input vector is $\begin{pmatrix} x \\ y \end{pmatrix}$. The next thing that happens is that this is passed through a thresholding operation. This is indicated by the sigmoid shape. There are various forms of thresholder; the so called *hard limiter* just takes the sign of the output, if *ax+by+c* is positive, the unit outputs 1, if negative or zero it outputs -1. Some people prefer 0 to -1, but this makes no essential difference to the operation of the net. As described, the function applied to *ax + by + c* is called the *sgn* function, not to be confused with the sine function, although they sound the same.

The network is, in some respects, easier to handle if the sigmoid function is smooth. A smooth approximation to the *sgn* function is easy to construct. The function *tanh* is sometimes favoured, defined by

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

If you don't like outputs which are in the range from -1 to 1 and want outputs which are in the range from 0 to 1, all you have to do is to add 1 and divide by 2. In the case of *tanh* this gives the sigmoid

$$sig(x) = \frac{e^x}{e^x + e^{-x}}$$

These sigmoids are sometimes called `squashing functions' in the neural net literature, presumably because they squash the output into a bounded range. In other books they are called *activation functions.*

We have, then, that the net of *Fig.1.6.* is a map from $\mathbb{R}^2$ to $\mathbb{R}$ given by

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightsquigarrow sig(ax + by + c)$$

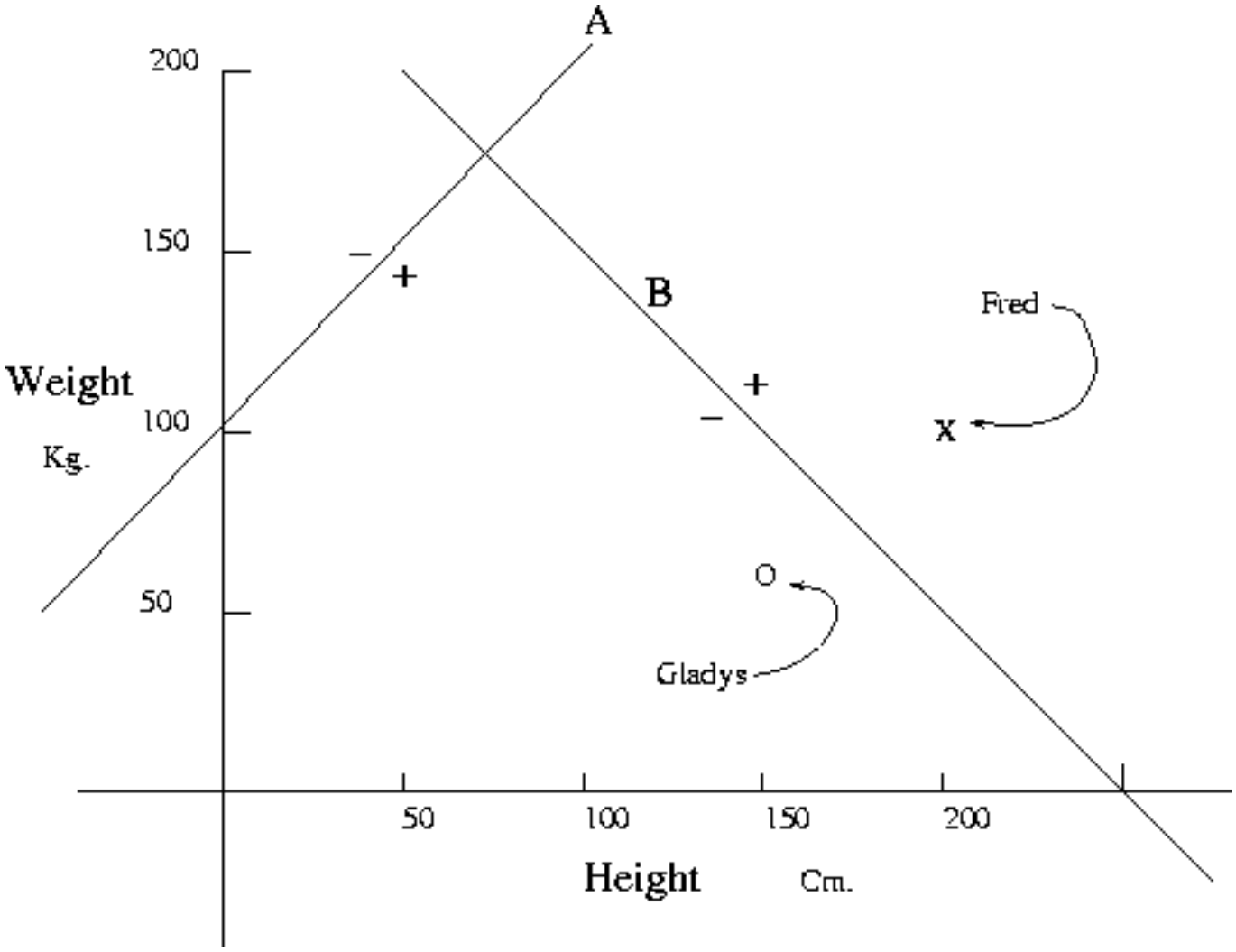
In the case where *sig* is just *sgn*, this map sends half the plane to the number 1 and the other half to the

number -1. The changeover occurs along the line $ax + by + c = 0$. It is common to think of the unit representing a neuron which `fires' if the weighted input $ax + by$ exceeds the threshold $-c$. This is picturesque, harmless and possibly inspirational. But it is more useful to say that the net divides up the plane $\mathbb{R}^2$ by a line, and one side of the line has points which get assigned to +1 and the other side of the line has all its points sent to -1.

So we can draw a line in the plane for any value of a, b and c, and attach a sign, + and -, to each side of the line. This is done in Fig.1.7, for two choices of the weights a,b and c.

With the choice $a = 1$, $b = -1$ and $c = 100$, we get the line labelled A, while with the choice $a = 1$, $b = 1$, and $c = -250$ we get the line labelled B. This last has the satisfactory property that it allows us to distinguish between Fred and Gladys, since Fred is taken to +1 and Gladys to -1 by the net. If you like, the neuron fires when Fred is input, but not when Gladys is input. We have a neuron which is capable of discrimination on the basis of sex, just like you and me.

Figure 1.7: Two possible states of the same neural net



It should now be clear that a neural net of the simple structure of *Fig.1.6.* cuts the plane into two halves by a dividing line. If the data set is as simple as *{ Fred, Gladys }* then the problem is to find out the right place to put the dividing line, that is, we have to find a choice of *a,b,c* that will put the points of one category on the positive side of the line and the points of the other category all on the other side. Our thinking about this simple case therefore leads us to three issues:

1. Can we find an algorithm for working out where to put the line in the plane?
2. Can we do the same sort of thing in higher dimensions?
3. What if the data set is like the male/female data of *Fig.1.2*? Can we put in more than one line so as to separate out the sets? It is obvious that a single line won't work.

The answers are gratifyingly positive (or I should not have asked the questions). The algorithm for a single unit as in the case of *Fig. 1.6.* is the well known *Perceptron Convergence Algorithm* and goes back to Rosenblatt and Widrow, among others, and will be described in later chapters. The dimension is largely irrelevant: if we had inputs *x, y* and *z* instead of *x* and *y*, we would have an extra weight and be looking at a separating surface with equation $ax + by + cz + d = 0$ which is the equation of a plane. In general, if $\mathbf{x}$ is a point in $\mathbb{R}^n$ let $\hat{\mathbf{x}}$ denote the point in $\mathbb{R}^{n+1}$ obtained by writing out the components of $\mathbf{x}$ and then putting a 1 in the last place. Then an element of the *weight space* for a single unit net with *n* inputs, i.e. the list of *n*+1 weights attached to the arcs going from the inputs (and threshold 1) to the unit can be regarded as a vector in $\mathbb{R}^{n+1}$, and if we call it $\mathbf{w}$, then the space $\mathbb{R}^n$ is divided into two halves by the hyperplane $\mathbf{w} \cdot \hat{\mathbf{x}} = 0$, where $\cdot$ denotes the standard inner or `dot' product. This is standard linear algebra, and should not trouble the well informed reader. If it boggles your mind, mind, then the remedy is to proceed to the Linear Algebra book obtainable from HeavenForBooks.com . You should rejoin the rest of us at this point after mastering the subject to the level of being able to understand Nilsson's book, mentioned in the bibliography at the end of this chapter.

The Perceptron Convergence Algorithm works for any dimension, as we shall see later. It takes some randomly chosen initial hyperplane and operates on it by selecting a data point, usually also at random, and then kicking the hyperplane around, then repeating for new, randomly selected points, until the hyperplane moves into the right position. This is called *training* the net. It isn't immediately obvious that there is such a rule for kicking hyperplanes around, but there is and it takes only some elementary linear algebra to find it. I shall explain all in a later chapter, but for now it suffices to get the general idea.

For more complicated data sets, we may need more than one unit in the second layer. For practical applications, it is also a good idea to have more than one layer; again this will be discussed in a later chapter. Training these more complicated nets so that they put many hyperplanes in reasonable positions is a little harder. This is what the *Back Propagation* algorithm accomplishes.

The purposes of this section will have been met if the reader understands that what an ANN does is to chop the space up into regions by means of hyperplanes, so that points of the same category are generally in the same regions. The decision as to where to put the dividing hyperplanes is taken by means of a *training algorithm* which usually means selecting data points and operating with them on a randomly

chosen initial placing of the hyperplanes until *convergence* has occurred or the *error measure* is small enough.

ANN's can take a long time to train, but they are often quite fast to use as pattern classifiers because we have to compute some number of inner products, a number usually much less than the amount of data. Chopping the space up into regions, each of which is, as we shall see later, convex, can be rationalised by observing that if a point is surrounded by points all of one category, with no points of another category in between, then it surely ought to be of the same category as its surrounding points in any reasonable universe. This is a weaker kind of assumption of reasonableness to make than the metric assumption, but whether the universe is prepared to go along with it has to be tested on particular data. It is easy to see that the hyperplane (line in dimension 2) B of *Fig.1.7.* which does an adequate job of telling Fred from Gladys is unlikely to keep on doing a good job of telling the guys from the gals as more data comes in. The hyperplane is a kind of *theory*. It has its opinions about the category of any new point that may be offered. A good theory has to be right when tested on new data, and the theory given by line B does not look promising. Another serious drawback of the ANN described by B is that an object weighing in at 50 Kg. and having a height of three metres is unequivocally theorised to be a man. Modifying the ANN so that it admits that it has never seen anything like it before and consequently doesn't have the foggiest idea what class a new point belongs to, is not particularly easy.

Next Up Previous Contents

Next: Statistical Methods **Up:** Decisions, decisions.. **Previous:** Metric Methods *Mike Alder 9/19/1997*

Statistical Methods

These fall into two broad types.

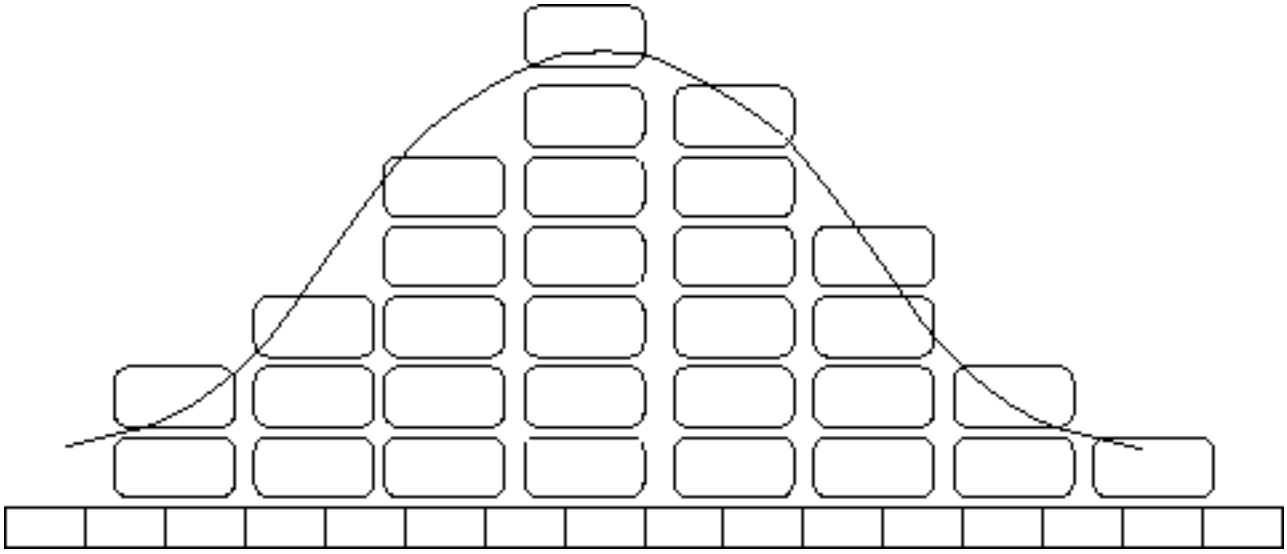
- Parametric
- Non-parametric

Mike Alder
9/19/1997

Parametric

Returning to the data set of the guys and the gals, you will, if you have had any amount of statistical education (and if you haven't, read up on Information Theory to acquire some), have immediately thought that the cluster of men looked very like what would be described by a *bivariate normal* or *gaussian* distribution, and that the cluster of women looked very like another. In elementary books introducing the one dimensional normal distribution, it is quite common to picture the distribution by getting people to stand with their backs to a wall, with people of the same height standing in front of each other. Then the curve passing through the people furthest from the wall is the familiar bell shaped one of *Fig.1.8.*, with its largest value at the average height of the sample.

Figure 1.8: One dimensional (univariate) *normal* or *gaussian* function



The function family for the one dimensional (univariate) gaussian distribution has two parameters, the centre, μ and the standard deviation, σ. Once these are assigned values, then the function is specified

(so long as σ is positive!) and of course we all know well the expression

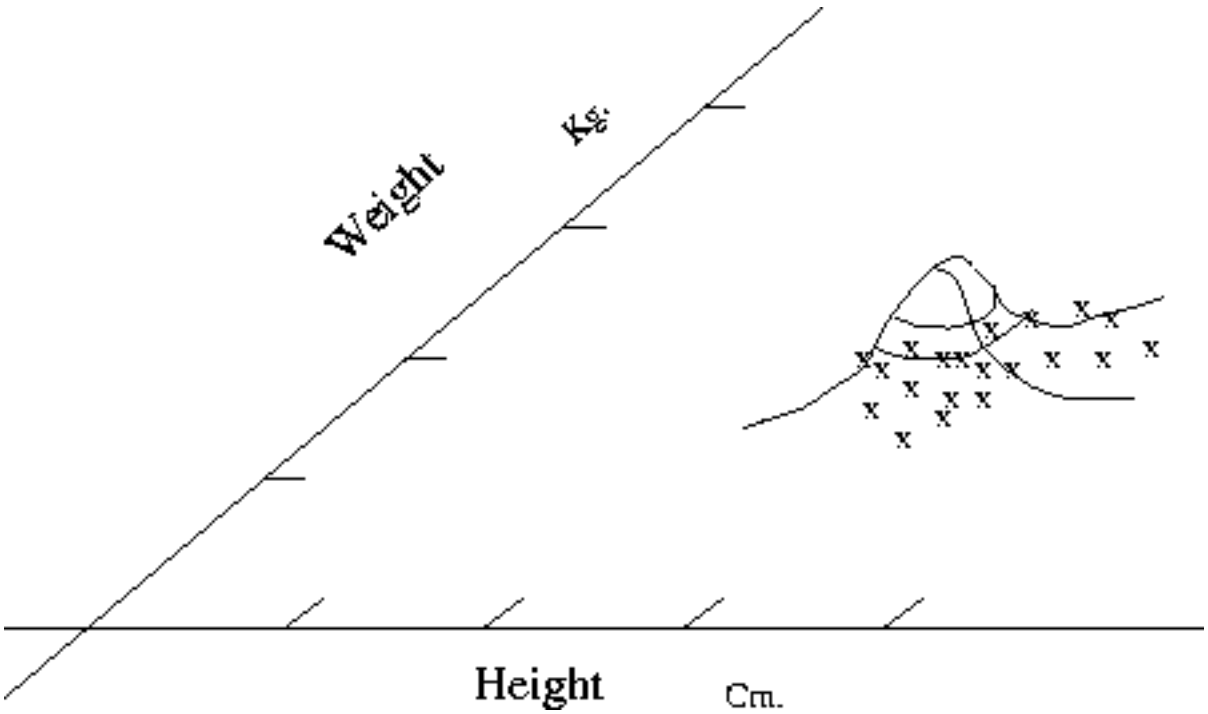
$$g_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

which describes the function algebraically.

The distribution of heights of a sample of men may be modelled approximately by the gaussian function in dimension 1 for suitably chosen values of μ, σ . The modelling process means that if you want an estimate of the proportion of the sample between, say, 170 and 190 cm. tall, it can be found by integrating the function between those values. The gaussian $g_{\mu,\sigma}$ takes only positive values, and the integral from $-\infty$ to ∞ is 1, so we are simply measuring the area under the curve between two vertical lines, one at 170 and the other at 190. It also follows that there is some fraction of the sample having heights between -50 and -12 cm. This should convince you of the risk of using models without due thought. In low dimensions, the thought is easy, in higher dimensions it may not be. To the philosopher, using a model known to be `wrong' is a kind of sin, but in statistics and probability modelling, we do not have the luxury of being given models which are `true', except possibly in very simple cases.

To visualise the data of men's heights and weights as modelled by a gaussian function in two dimensions, we need to imagine a `gaussian hill' sitting over the data, as sketched rather amateurishly in *Fig.1 .9.* Don't shoot the author, he's doing his best.

Figure 1.9: Two dimensional (bivariate) normal or gaussian distribution



This time the gaussian function is of two variables, say *x* and *y*, and its parameters now are more complicated. The centre, μ is now a point in the space $\mathbb{R}^2$, while the σ has become changed rather

more radically. Casting your mind back to your elementary linear algebra education, you will recall that quadratic functions of two variables may be conveniently represented by symmetric matrices, for example the function

$$f : \mathbb{R}^2 \longrightarrow \mathbb{R}$$

given by

$$f\begin{pmatrix} x \\ y \end{pmatrix} = 3x^2 + 8xy + 7y^2$$

may be represented by the matrix

$$\begin{pmatrix} 3 & 4 \\ 4 & 7 \end{pmatrix}$$

and in general for quadratic functions of two variables we can write

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightsquigarrow \begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

for the function usually written $ax^2 + 2bxy + cy^2$. Multiplying out the matrices gives the correct result.

Since in one dimension the gaussian function exponentiates a quadratic form, it is no surprise that it does the same in two or more dimensions. The n-dimensional gaussian family is parametrised by a centre, $\mathbf{m}$ which is a point in $\mathbb{R}^n$ and $\mathbf{V}$ which is an n by n invertible positive definite symmetric matrix representing the quadratic map which takes $\mathbf{x}$ to $\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x}$. The symbol ^{T} denotes the transpose of the column matrix to a row matrix. The formula for a gaussian function is therefore

$$g_{[\mathbf{m},\mathbf{V}]}(\mathbf{x}) = \frac{1}{(\sqrt{2\pi})^n \left(\sqrt{det(\mathbf{V})}\right)} e^{-\frac{(\mathbf{x}-\mathbf{m})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{m})}{2}}$$

and we shall refer to $\mathbf{m}$ as the *centre* of the gaussian and $\mathbf{V}$ as its *covariance matrix*. The normal or gaussian function with centre $\mathbf{m}$ and covariance matrix $\mathbf{V}$ is often written N($\mathbf{m}$, $\mathbf{V}$) for short. All this may be found explained and justified, to some extent, in the undergraduate textbooks on statistics. See Feller, *An Introduction to Probability Theory and Applications* volume 2, John Wiley 1971, for a rather old fashioned treatment. Go to Information Theory for a more modern explanation.

The parameters $\mathbf{m}$ and $\mathbf{V}$ when given actual numerical values determine just one gaussian hill, but we

have the problem of working out which of the numerical values to select. The parameter space of possible values allows an infinite family of possible gaussian hills. If we believe that there is some suitable choice of $\mathbf{m}$ and $\mathbf{V}$ which will give, of all possible choices, the *best fit* gaussian hill to the data of *Fig.1.9.*, then we can rely on the statisticians to have found a way of calculating it from the data. We shall go into this matter in more depth later, but indeed the statisticians have been diligent and algorithms exist for computing a suitable $\mathbf{m}$ and $\mathbf{V}$. These will, in effect, give a function the graph of which is a gaussian hill sitting over the points. And the same algorithms applied to the female data points of *Fig.1.2.* will give a second gaussian hill sitting over the female points. The two hills will intersect in some curve, but we shall imagine each of them sitting in place over their respective data points- and also over each others. Let us call them g_m and g_f for the male and female gaussian functions respectively.

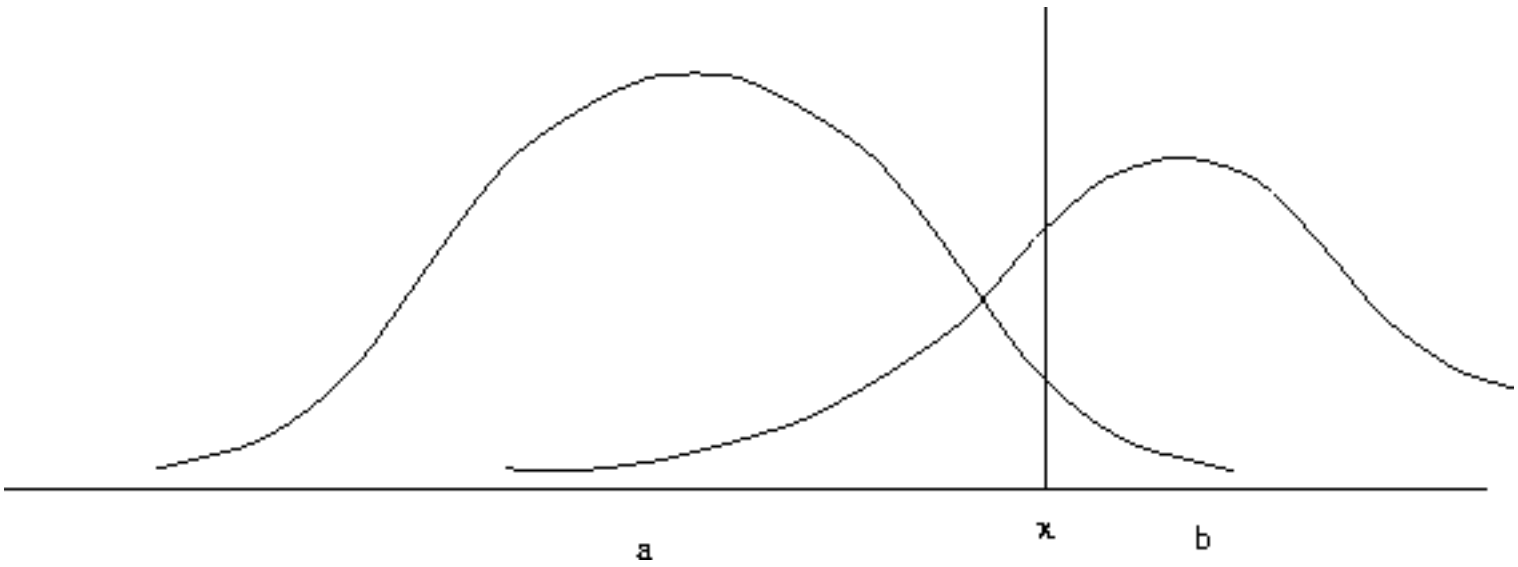
If a new data point $\mathbf{x}$ is provided, we can calculate the height of the two hills at that point, $g_m(\mathbf{x})$ and $g_f(\mathbf{x})$ respectively. It is intuitively appealing to argue that if the male hill is higher than the female hill at the new point, then it is more likely that the new point is male than female. Indeed, we can say how much more likely by looking at the ratio of the two numbers, the so called *likelihood ratio*

$$\frac{g_m(\mathbf{x})}{g_f(\mathbf{x})}$$

Moreover, we can fairly easily tell if a point is a long way from any data we have seen before because both the *likelihoods* $g_m(\mathbf{x})$ and $g_f(\mathbf{x})$ will be small. What `small' means is going to depend on the dimension, but not on the data.

It is somewhat easier to visualise this in the one dimensional case: *Fig.1.10.* shows a new point, and the two gaussian functions sitting over it; the argument that says it is more likely to belong to the function giving the greater height may be quantified and made more respectable, but is intuitively appealing. The (relatively) respectable version of this is called *Bayesian Decision Theory*, and will be described properly later.

Figure 1.10: Two gaussian distributions over a point of unknown type.



The advantage of the parametric statistical approach is that we have an explicit (statistical) model of a process by which the data was generated. In this case, we imagine that the data points were generated by a process which can keep producing new points. In *Fig.1.10.* one can imagine that two darts players of different degrees of inebriation are throwing darts at a line. One is aiming at the centre *a* and the other, somewhat drunker, at the centre *b*. The two distributions tell you something about the way the players are likely to place the darts; then we ask, for the new point, *x*, what is the probability that it was thrown by each of the two players? If the *b* curve is twice the height of the *a* curve over *x*, then if all other things were equal, we should be inclined to think it twice as likely that it was aimed by the *b* player than the *a*.

We do not usually believe in the existence of inebriated darts players as the source of the data, but we do suppose that the data is generated in much the same way; there is an ideal centre which is, so to speak, aimed at, and in various directions, different amounts of scatter can be expected. In the case of height and weight, we imagine that when mother nature, god, allah or the blind forces of evolution designed human beings, there is some height and weight and shape for each sex which is most likely to occur, and lots of factors of a genetic and environmental sort which militate in one direction or another for a particular individual. Seeing mother nature as throwing a drunken dart instead of casting some genetic dice is, after all, merely a more geometric metaphor.

Whenever we make a stab at guessing which is the more likely source of a given data point coding an object, or alternatively making a decision as to which category an object belongs, we have some kind of tacit model of the production process or at least some of its properties. In the metric method, we postulate that the metric on the space is a measure of similarity of the objects, in the neural net method we postulate that at least some sort of convexity property holds for the generating process.

Note that in the case of the statistical model, something like the relevant metric to use is generated automatically, so the problem of *Fig.1.4.* is solved by the calculation of the two gaussians (and the X-axis gets shrunk, in effect). The rationale is rather dependent on the choice of gaussians to model the data. In the case discussed, of heights and weights of human beings, it looks fairly plausible, up to a point, but it may be rather difficult to tell if it is reasonable in higher dimensions. Also, it is not altogether clear what to do when the data does not look as if a gaussian model is appropriate. Parametric models have been used, subject to these reservations, for some centuries, and undoubtedly have their uses. There are techniques in existence for coping with the problems of non-gaussian distributions of

data, and some will be discussed later. The (Bayesian) use of the likelihood ratio to select the best bet has its own rationale, which can extend to the case where we have some prior expectations about which category is most likely. Again, we shall return to this in more detail later.

Next Up Previous Contents

Next: Non-parametric **Up:** Statistical Methods **Previous:** Statistical Methods *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: CART et al **Up:** Statistical Methods **Previous:** Parametric

Non-parametric

Suppose we assume that there is some *probability density function* (*pdf* for short) for the men and another for the women, but we are unwilling to give a commitment to gaussians or any other family of functions to represent them. About the weakest condition we might apply is that the two pdf's are continuous. We could try to estimate them locally, or at least the likelihood ratio, in a neighbourhood of the new datum. One way of doing this is to take a ball in the space centred on the new point. Now count the number of points in each category that are within the ball. The ratio of these two numbers is our estimate of the likelihood ratio. These are called *Parzen* estimates.

Of course, one number or the other might easily be zero if the ball is too small, but if the ball is too big it might measure only the total number of points in each category. Oh well, life wasn't meant to be easy.

An alternative is to take, for some positive integer k, the k nearest neighbours of the new point, and count those in each category. Again, the bigger number is the best guess. This is called the *k nearest neighbours* algorithm. It looks rather like the metric method with which we started the quest to get sensible answers to the pattern recognition problem, but is making different assumptions about the nature of the process producing the data. Again, there is a problem of how to measure the distances. In either alternative, it is possible to weight the count of points inversely by distance from the place of interest, so that remote points count less than close ones. This brings us back, yet again, to the question of what the right metric is.

Some people have argued that Artificial Neural Nets are just a non-parametric statistical method of making decisions: this is debatable but not profitably.

Next Up Previous Contents

Next: CART et al **Up:** Statistical Methods **Previous:** Parametric *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Clustering: supervised v unsupervised **Up:** Decisions, decisions.. **Previous:** Non-parametric

CART et al

There is an approach to the problem of which I shall have little to say, although it has its proponents and its merits. It is typified by CART, and it works roughly as follows.

Suppose we want to tell the gals from the guys again. We take the two dimensional weight and height representation for illustration.

We first see how to cut the space up into two sections by working out the best place to put a hyperplane (line) in the space so as to get the largest fraction of points correctly discriminated. This is just like the MLP with a single unit so far.

Then we look at the points that are wrong, and try to fix them up by further subdivision of the space. We repeat until we have everything right, or satisfy some other, less exacting, criterion. We wind up, typically, with a tree structure to decide what the category of a point is, going through a sequence of binary decisions.

This approximates what a Multi-Layer Perceptron with two hidden layers accomplishes, although the algorithms are generally faster and more intelligent.

The scope for generalising is obvious; for example we do not need the data points to be all in the same space, since we can classify on, for example, the dimension. We may have a mix of geometric and symbolic representations, and so on.

My reason for not expanding on this area is because I am not happy with either the range of possible representation systems, or the segmentation process. I think there are better ways to do it, and I shall indicate them later in this book.

Next Up Previous Contents

Next: Clustering: supervised v unsupervised **Up:** Decisions, decisions.. **Previous:** Non-parametric *Mike Alder*
9/19/1997

Clustering: supervised v unsupervised learning

The reflective reader will, perhaps, have been turning to the not so silly question of how he or she tells men from women. Or to put it another way, looking at the clusters of points in *Fig.1.2.*, if instead of having labelled one set as X points for males and O points for females, suppose we had just drawn unlabelled points as little black dots: could a program have looked at the data and seen that there are two populations? It seems reasonable to suppose that the reader, with somewhat different sensory apparatus, has some internal way of representing human beings via neurons, or in wetware as we say in the trade, and that this shares with the capacity for coding resemblance or similarity in terms of proximity. Then the dimension may be a little higher for you, dear reader, but most of the problem survives.

There is, undeniably, a certain amount of overlap between the two clusters when we measure the weight and height, and indeed there would be *some* overlap on any system of measurement. It is still the case however (despite the lobbying of those who for various reasons prefer not to be assigned an unambiguous sex) that it is possible to find measurement processes which do lead to fairly well defined clusters. A count of X and Y chromosomes for example. Given two such clusters, the existence of the categories more or less follows.

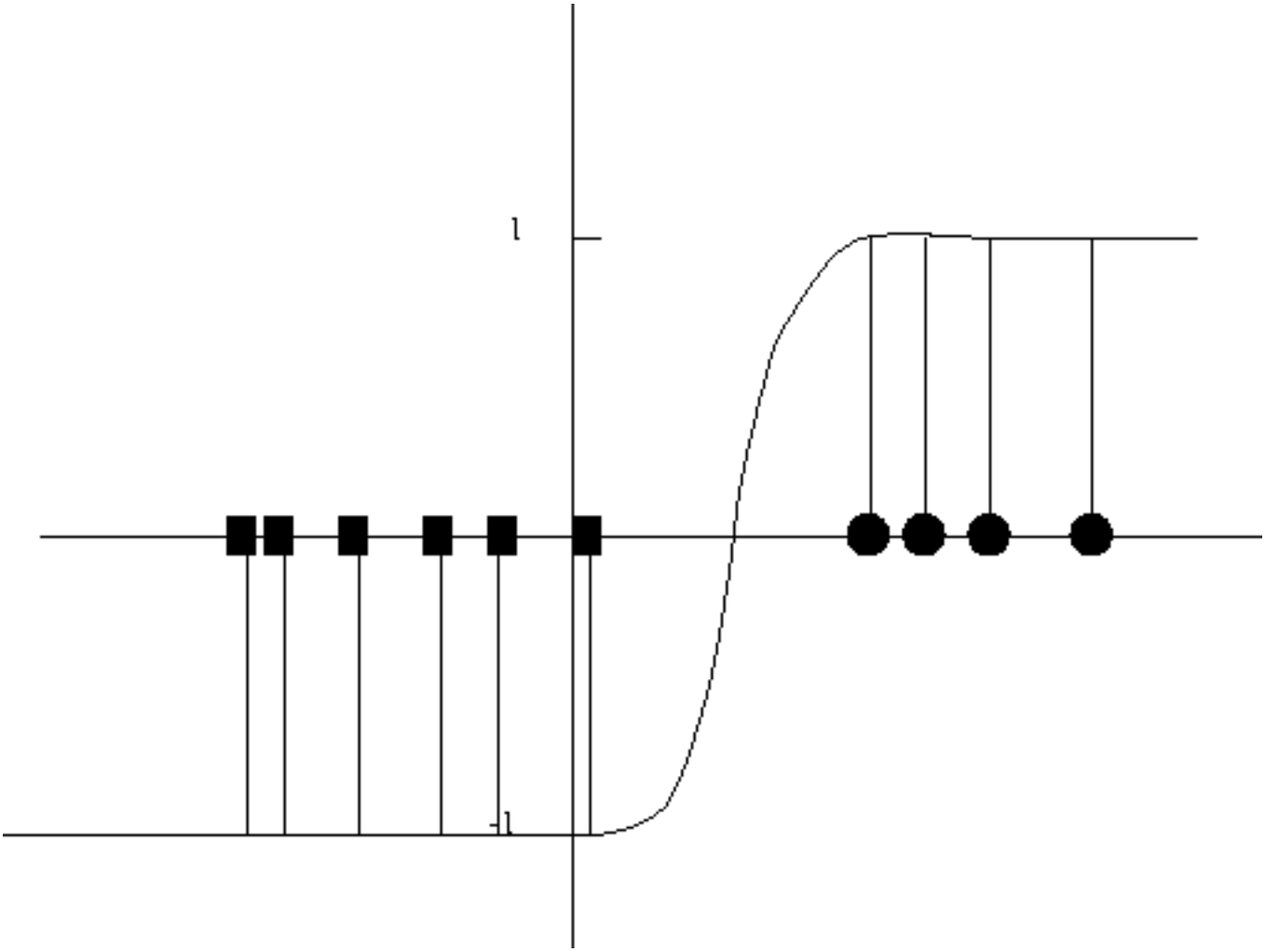
One of the reasons for being unhappy with the neural net model we have described is that it is crucially dependent on the classification being given by some external agent. It would be nice if we had a system which could actually *learn* the fact that women and men are distinguishable categories by simply noticing that the data form two clusters.

It has to be assumed that at some point human beings learn to classify without any immediate feedback from an external agent. Of course, kicking a neuron when it is wrong about a classification, and kicking a dog when it digs up the roses have much the same effect; the devastation is classified as `bad' in the mind of the dog, or at least, the owner of the rose bush hopes so. It is not too far fetched to imagine that there are some pain receptors which act on neurons responsible for classifying experiences as `good' and `bad' in a manner essentially similar to what happens in neural nets. But most learning is a more subtle matter than this; a sea anemone `learns' when the tide is coming in without getting a kick in the metaphorical pants. Mistaking a man for a woman or *vice versa* might be embarrassing, but it is hard to believe you learnt the difference between men and women by making many errors and then reducing the average embarrassment, which is how an artificial neuron of the classical type would do it.

Learning a value, a +1 or -1 for each point in some set of points, and then being asked to produce a rule or algorithm for guessing the value at some new point, is most usefully thought of as fitting a function to a space when we know its value on a finite data set. The function in this case can take only binary values,

± 1 , but this is not in principle different from drawing a smooth curve (or surface) through a set of points. The diagram *Fig.1.11.* makes it clear, in one dimension, that we are just fitting a function to data.

Figure 1.11: A one dimensional pattern recognition problem solved by a neural net.



This perspective can be applied to the use of nets in control theory applications, where they are used to learn functions which are not just binary valued.

So *Supervised Learning* is function fitting, while *Unsupervised Learning* is cluster finding. Both are important things to be able to do, and we shall be investigating them throughout this book.

Next Up Previous Contents

Next: Dynamic Patterns **Up:** Basic Concepts **Previous:** CART et al *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Structured Patterns **Up:** Basic Concepts **Previous:** Clustering: supervised v unsupervised

Dynamic Patterns

The above classification of learning systems into *supervised* and *unsupervised* , function fitting and clustering, although lacking in formal precision, is of some intuitive value. We are, of course, conducting a leisurely survey of the basic concepts at present, rather than getting down to the nitty-gritty and the computational, because it is much easier to get the sums right when you can see what they are trying to accomplish.

The framework discussed so far, however, has concentrated on recognising things which just sit there and wait to be recognised; but many things change in time in distinctive ways. As an example, if we record the position and possibly the pressure of a stylus on a pad, we can try to work out what characters are being written when the user writes a memo to himself. This gives us a trajectory in dimension two to classify. Or we might have an image of a butterfly and a bird captured on videotape, and wish to identify them, or, more pressingly, two kinds of aeroplane or missile to distinguish. In these cases, we have trajectories in $\mathbb{R}^2$ or possibly $\mathbb{R}^3$ as the objects to be recognised. A similar situation occurs when we recognise speech, or try to: the first thing that is done is to take the time sequence which gives the microphone output as a function of time and to perform some kind of analysis of its component frequencies, either by a hardware filter bank, an FFT (Fast Fourier Transform) followed by some binning so as to give a software simulation of the hardware filterbank, or relatively exotic methods such as Cepstral Coefficients or Linear Predictive Coding coefficients. All of these transform the utterance into a trajectory in some space $\mathbb{R}^n$ for n anywhere between 2 and 256. Distinguishing the word `yes' from the word `no', is then essentially similar to telling butterflies from birds, or boeings from baseballs, on the basis of their trajectory characteristics.

An even more primitive problem occurs when one is given a string of ascii characters and has to assign provenance. For example, if I give you a large sample of Shakespearean text and a sample of Marlowe's writing, and then ask you to tell me what category does a piece written by Bacon come under, either or neither, then I am asking for a classification of sequences of symbols. One of the standard methods of doing Speech Recognition consists of chopping up the space of speech sounds into lumps (A process called *vector quantisation* in the official documents) and labelling each lump with a symbol. Then an utterance gets turned first into a trajectory through the space, and then into a sequence of symbols, as we trace to see what lump the trajectory is in at different times. Then we try to classify the symbol strings. This might seem, to the naive, a bizarre approach, but it might sound more impressive if we spoke of vector quantisation and Hidden Markov Models. In this form, it is more or less a staple of speech recognition, and is coming into favour in other forms of trajectory analysis.

The classification of trajectories, either in $\mathbb{R}^n$ or in some discrete alphabet space, will also therefore preoccupy us at later stages. Much work has been done on these in various areas: engineers wanting to

clean up signals have developed adaptive filters which have to learn properties of the signal as the signal is transmitted, and statisticians and physicists have studied ways to clean up dirty pictures. Bayesian methods of updating models as data is acquired, look very like skeletal models for learning, and we shall be interested in the extent to which we can use these ideas, because learning and adaption are very much things that brains do, and are a part of getting to be better at recognising and classifying and, in the case of trajectories, predicting.

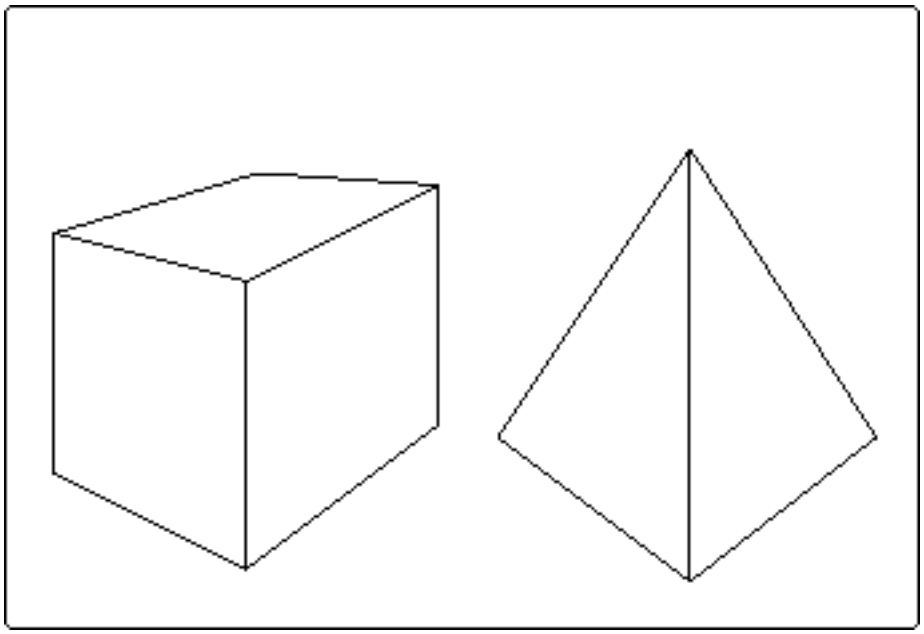
Next Up Previous Contents

Next: Structured Patterns **Up:** Basic Concepts **Previous:** Clustering: supervised v unsupervised *Mike Alder*
9/19/1997

Structured Patterns

The possibility that instead of having a single point in $\mathbb{R}^n$ to classify we shall have a trajectory is only the tip of an iceberg. The temporal order is about the simplest that can be imposed on a set of points in $\mathbb{R}^n$, but it is far from being the only one.

Figure 1.12: Structured objects in an image



To see another possibility, contemplate the problem of recognising an image of a line drawing of a cube and distinguishing it from an image of a line drawing of a pyramid. The approach suggested so far would be to find some measurement operation on the image which would do the job. This is obviously possible. If we were to count edges in some way, that would solve the problem without even having to worry about which edges joined to which.

The trouble is, it requires the pattern recognising human to choose, for each geometrical object, some measurement process specific to the objects to be recognised. This is currently how things are done; when somebody writes a program to recognise chinese characters, he sits and thinks for a while about how to make some measurements on them so as to give some resulting point in $\mathbb{R}^n$ for each character, or if not a point in $\mathbb{R}^n$, some other kind of representation. Having done this for the kinds of objects he is

interested in classifying, he then tries to automate the process of producing the point or other symbolic representation describing the original object, and then he sets about writing a program to classify the representations.

The process which chooses the representation is in the head of the programmer, the program does not make the decision for him. This makes a lot of pattern recognition rather slow; it provides employment for any number of hackers, of course, which is something hackers all think is a good thing, but it plainly isn't particularly satisfactory to the thinkers among us. It looks to be something which can be and ought to be automated; the approach would have to be concerned with extracting some information about how parts of the object are built up out of sub-parts, and suitably coding this information.

A related issue is the *scene analysis* problem, when one image contains several subimages each of which is required to be recognised. In Optical Character Recognition (OCR) for instance, one is usually given a page with rather a lot of characters on it, and the first task is usually to segment the image into bits. This can be very difficult when the objects touch. A photograph of your dear old Grandmother in front of the house does not usually stop you recognising both granny and the house.

One might hope that it is possible to say how some pixels aggregate to form lines or edges, some edges aggregate to form corners, some corners aggregate to form faces, and some faces aggregate to form a cube, or possibly a pyramid. Similarly, an image of an aeroplane is made up out of subimages of tail, wings and fuselage; an image of a face is made up out of a nose, eyes and a mouth.

The arrangement of the bits is crucial, and is easily learnt by quite small children. Counting the number of people grinning out at from a photograph is easy for the small child. The hacker confronted with a problem like this usually counts eyes and divides by two, getting the wrong answer if there are grapes in the picture, or if Aunty Eth is hidden behind Uncle Bert except for the hat. His program can be thrown off by the shine on someone's spectacles. When it works, it has been developed until it contains quite a lot of information about what the programmer thinks faces are like. This can take a long time to get into the machine, and it can be wrong. It would be necessary to start all over again if you wanted to count lumps of gravel or blood cells. It is clear that human beings don't have to learn by being told everything in this way; they can figure out things for themselves. It would be nice if our programs could do the same, if only in a small way. This can in fact be done, by methods which may perhaps mimic, abstractly, the central nervous system, and I shall describe them in later chapters under the heading of *Syntactic Pattern Recognition*.

Next Up Previous Contents

Next: Alternative Representations **Up:** Basic Concepts **Previous:** Dynamic Patterns *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Strings, propositions, predicates and **Up:** Basic Concepts **Previous:** Structured Patterns

Alternative Representations

In this slow and gentle walk through the basic ideas of Pattern Recognition, I have concentrated on representing objects by some kind of measuring process which translates them into points in a vector space. There are a few other methods of representation which must be mentioned.

The vector space method of coding a description of the state of affairs runs right through science and is deeply embedded in the psyche of anyone who has survived an undergraduate course in Physics or Engineering. On the other hand, it seems bizarre and unnatural to, say, psychologists or computer scientists.

The power of the system of representation may be judged by what has been accomplished by it: virtually all of modern technology. The biological scientists might think otherwise, but almost all their measuring devices have been supplied by engineers working along lines worked out by physicists and chemists. Without such devices as x-ray diffraction systems and centrifuges, and an understanding of electrophoresis and isotopes, genetics would still be horse and flower breeding. This may sound a strong statement, and indeed it is, but some extended reflection is likely to convince the informed reader of its truth. The spectrum of ideas upon which contemporary science and technology depend, from statistics to electromagnetism, quantum mechanics to fluid mechanics, geology to developmental morphology, very much depends upon describing a system by means of a real or complex vector, sometimes an infinite dimensional one otherwise known as a function. There is a natural prejudice in favour of this representation language in almost all the practioners who come at the subject from a conventional science or engineering background, to the point where it may not occur to them that there is any sane alternative. The success of the language affords sufficient justification for using it, but the reasonable man will want to satisfy himself that the alternatives are not inherently superior.

- Strings, propositions, predicates and logic
- Fuzzy Thinking
- Robots

Next Up Previous Contents

Next: Strings, propositions, predicates and **Up:** Basic Concepts **Previous:** Structured Patterns *Mike Alder*
9/19/1997

Strings, propositions, predicates and logic

Philosophers have been known to get quite indignant about coding information as vectors of real numbers. They point out, correctly, that there is much, much more to the physical system than turns up in the description. This, they asseverate, is even more true for any attempt to describe something as subtle and difficult as human thought processes. They usually stop at this point, waiting for an attempt at rebuttal.

Computer scientists committed to *Artificial Intelligence (AI)* tend to use symbol strings to describe objects. Well, a vector is a symbol string too, of course, but the AI worker tends to prefer alphabetic strings of varying length. A natural language such as English is an example of such a system. This system, the only one most philosophers know how to use, also abstracts a pathetically small amount of information concerning the system described. The definitive poem about love which says all there is to say, has not yet been written and is unlikely to be short. Since poets and other literary men have not illuminated the rest of us very effectively on the details of how to build any complex system, we may conclude that natural language has its strengths in other areas. It works well for asking people to pass the salt or telling them you love them, the latter being more an expression of an internal state than a proposition having verifiable content, but in the main it simply codes our ignorance in ways we don't understand, while a mathematical theory codes our ignorance in ways which we *do*, to some extent, understand. The computer language LISP, much favoured by the *soi disant* Artificial Intelligentsia, is another language which allows easy coding of information about objects in terms of symbol strings. LISP is about half way between natural language and Mathematics in terms of precision and scope, so the results of coding information in LISP strings usually results in the worst of both worlds.

To give an example of what we might accomplish with string representations of data, imagine that we have got a coding of each individual in the data set of the guys and the gals from *Fig.1.2.*, so that instead of having him or her represented by two numbers we have a *list* associated with him or her. One such individual might have the list:

$$\left(\begin{array}{c} name : Jim\,Jones \\ weight(kg) : 90 \\ height(cm) : 175 \\ favourite\ lipstick\ colour : none \\ preferred\ apparel : jeans \\ shoe\ size : 9 \\ sex : m \\ . \\ . \end{array}\right)$$

Similar lists, let us suppose, comprise the rest of the data, all of which describe either men or women. Now the job of working out the category is simply a matter of finding the seventh item on the list and scanning the string until finding the colon,`:'. The next symbol should be either an `m' or an `f'. This solves the problem.

But does it? What if the data were obtained by asking people to fill in questionnaires about themselves, and someone put `yes please' in answer to question 7? Or the respondent was firmly of the opinion that it was none of the interviewers business and said so? Well, we could go to the name in item 1 and look up the first name in a dictionary of names, sorted by sex. Unless the person is Chinese in which case it is the last name. Or we could argue that men seldom wear dresses, hate lipstick and have bigger feet than women. In general, some application of quasi-logical rules to the strings is required in order to come out with a conclusion. We may have to bear in mind that the respondents can (a) tell lies, (b) decline to answer some questions or (c) have idiosyncratic views about the right answers to the questions. Even if the lists have been compiled by some other person, the items may contain unexpected anomalies, like Jay Spondulix who likes coral lipstick and has preferred apparel `none'. Is Jay a man who likes his women with coral lipstick but otherwise nude, a woman who has no particular preference for what clothes she wears, or some combination? The fact that the procedure by which we obtain the list in the first place is so unreliable and that the meanings are ambiguous and vague, means that algorithms applied may easily produce nonsense.

Physical systems, by contrast, have the measuring processes much more tightly specified. It is true that the process of obtaining weights and heights may also have been carried out badly, but there are methods for determining weights and heights which are fairly well defined, widely known and reliable. Doing them again on a different day usually gives results which agree fairly well, and when they don't, we normally feel justified in saying that the system has changed between measurements.

The question of how one obtains the description is of particular importance in automation tasks. There are programs which decide if something is a bridge made of blocks by examining the _is_next_to_ and _is_on_top_of_ relations between the blocks, but the question of how one gets such data in the first place is not addressed. And all artificial sensors such as cameras, microphones and strain-gauges produce output which can be coded as a vector of real numbers, and there is a relatively simple relation between the equipment, the world, and the values.

To be fair, well, fairer, to the Artificial Intelligentsia (their term, not mine), it has to be conceded that they have become aware of the importance of choosing powerful representations of data. The trouble is, they don't seem to know any. (This is just one more of the provocative statements I warned you about in the introduction!)

Imagine two people sitting in large cardboard boxes and unable to see outside, but able to hear a person reading a book to them. Person A in box number 1 is equipped with a long, thin roll of paper and a pencil. Person B in box 2 has a few pounds of modelling clay in different colours, a bottle of little flags on cocktail sticks, a few marker pens and some blocks of wood. Person C reading the book has a loud clear voice and a chair to sit on.

Person C reads out a description of the environs of, say, Mansfield Park, complete with lake, river, hills and houses. While this is done, A is scribbling the results down in sentences, while B is building a model out of clay and blocks and flags. Now suppose that there is an inconsistency in C's story. At one point, one statement is made about the geography and at another point a logically incompatible statement is made, such as asserting that a lake both is and is not in a given location.

For A to detect this, assuming he has not recalled it all in his head, he must continually go through the entire set of sentences obtained at each time and do inferences from them. Deducing the inconsistency may take a long time. For B, the detection is immediate; the inference is done by the modelling clay. And if sentences are given which are not logically inconsistent but are physically impossible, say that a stream runs uphill somewhere, in order for A to detect this, the sentences have to be augmented by lots of other sentences of naive physics, and the logic of the whole lot checked. It is barely possible that this could be done. While B, with his modelling clay, can detect such physical anomalies the instant they are asserted by C.

We conclude from this that one man's inference engine is another man's modelling clay, or more generally that what is done by logical processes under a sentential representation of data can be accomplished by other means very much more efficiently given a more powerful representation system. The most powerful general symbolic representation schemes have been developed for the physical sciences and involve measurements and hence representations of systems by vectors. They code for similarity or proximity. The language of functions can allow us to describe densities of data, and to measure the amount of surprise we get when something unexpected happens. By contrast, logic and computer languages are barbarous, uncouth and primitive. Only someone ignorant of better linguistic tools would tolerate them.

Compared with the sophistication of mathematical systems which have been devised over the last few thousand years, logic is a low level language, unsuited for dealing with the complexities of the real world. To be constrained to using logic is analogous to having to program in machine code. And logic, or variants of it with little more power, are all that the string representation enthusiasts have.

Next Up Previous Contents

Next: Fuzzy Thinking **Up:** Alternative Representations **Previous:** Alternative Representations *Mike Alder*
9/19/1997

Fuzzy Thinking

Finally, in recent times some claims have been made for Fuzzy sets. Zadeh originally proposed Fuzzy Logic and Fuzzy sets as a model for imprecision in natural language. The idea was that whereas using classical logic we have that `Fred is a man', is interpreted as ` Fred is a member of the set of all men', the similar sounding proposition `Fred is a tall man' has a rather dubious interpretation as `Fred is a member of the set of tall men'. When is a man tall? Fuzzy set theory attempts to overcome the essential vagueness about how tall a tall man is by assigning a number between 0 and 1 to the proposition `Fred is a tall man', and hence assigns a number to the extent to which Fred is a member of the `Fuzzy set' of tall men. How this number is arrived at is never made clear. The vector space representation would simply measure his height.

Modelling objects by Fuzzy predicates certainly seems more useful than modelling them by the quasi-logical predicates of AI, and the area has grown in recent times. It is liable, however, to a serious criticism which has never been convincingly rebutted: everything fuzzy sets can do, statistics and probability theory can do, too. And statistics and probability theory have relatively well argued rationales, a huge body of successful applications and fairly solid foundations. In contrast, the so-called applications of Fuzzy Logic and Fuzzy Set theory are often just examples of fuzzy thinking. Giving a rather badly drawn oval shape as an example of something which has fuzzy membership in the set of ellipses, for example, suggests a misunderstanding of how descriptions in Science and Engineering work. They aren't there to give you a warm inner glow, they are there to be used. The question is, what can you do with such a statement? On all the evidence, not much. To see that these are issues in linguistics rather than science, consider the response of a Trobriand Islander to the same badly drawn oval shape. He might, just possibly, see it as a `fuzzy' coconut or a `fuzzy' necklace, but he wouldn't feel inclined to assign it as having some (hard to measure) degree of elementhood in the set of ellipses, if only because he hasn't met the set before. If you feel an urge to put a metric on the set of simple closed curves in the plane so as to measure the extent to which such things are ellipses, it is not too difficult to do, but it would be a mistake to imagine there is anything unique about your choice of how to do it, or that your urge requires expression just because you have it. I frequently have the urge to beat Fuzzy Set theorists over the head with a bottle, but I curb this impulse.

Early applications of these ideas to control have been unconvincing. An early but typical one consisted of replacing a thermostat system which monitored the temperature and fed power to a furnace so as to heat the place up if the temperature went low, and cool it down when it went high. The original controller measured the temperature continuously with a thermocouple, the `fuzzy' controller divided the temperatures into three intervals, `too hot', `too cold', and `OK'. Describing a controller which treats temperatures which fall into intervals as *fuzzy* is sharp practice. If the controller sometimes acted as though `too hot' started at 80^o and sometimes at 70^o, or if the thermocouple was subject to large random variations, there might be a case, but the world isn't fuzzy except possibly at the quantum level. What is fuzzy is language, and we are more interested in the world. So far as I am aware, the sudden surge of so

called AI applications of fuzzy logic in Japanese refrigerators is composed of ideas as banal as the fuzzy controller, and is intended to sell more refrigerators to the unwashed. Science and Engineering are rather more difficult than that.

Later applications of `Fuzzy Control' appear to be more interesting. The fuzzy control of the inverted pendulum, and more impressively the double inverted pendulum (balancing two billiard cues, one on top of the other) are claimed to work, whereas classical control theory finds these problems difficult. There is some reason to believe that the fuzzy control people have a good idea in there somewhere, but it seems doubtful if it is coherently worked out. Engineers and Physicists have a tradition of coming up with working systems using a rationale that is full of holes and logical nonsense. This has often led to mathematical advances when the mathematicians woke up to the fact that there had to be *something* there, and that the fact that it couldn't be expressed in conventional ways meant that mathematicians had to do some work. This may be the case with `fuzzy control'; they may have some good ideas mixed up with some awful philosophy and scrofulous mathematics that would be well worth sorting out properly.

Multi-valued logics have been around since Post in the 1930's, and John Maynard Keynes wrote around the same time on Probability interpretations in a way similar to Zadeh's model. More recently Ed Jaynes has given some good reasons for believing that the assignment of numbers to statements to measure their believability must be done in accordance with Bayesian statistics. This is discussed in the Information Theory literature.

What it comes down to is that when we have uncertainty in the real world, the semantics of our models involve counting things that happen in different outcomes when we cannot distinguish between the inputs, as when we throw dice. This is what probability theory was invented for, and it seems to do a reasonable job. When we wish to treat of such notions as closeness or proximity, we have all the machinery of topological and metric spaces already to hand. It is pointless to reinvent it rather amateurishly when it has been done well once already. Reinventing wheels is sometimes justifiable, but reinventing statistics and topology, badly, is time and energy wasted.

Fuzzy set theory seems to have been based upon and be driven by a desire to translate vague sentences of natural language into a mathematical formalism. This seems like a great idea to those many people who cannot distinguish clearly between properties of the world and properties of the language we use to talk about it. Such people may genuinely believe that the world is fuzzy around the edges.

Possibly for them it is; a consequence, perhaps, of having smoked or sniffed illegal substances as undergraduates. They should have stuck to brandy, cigars and sex, like the rest of us.

Getting clear descriptions of what is going on in a nuclear power plant is not much helped by going and asking the man in the street what he thinks. His description will certainly be describable as `fuzzy'. But most engineers would swap it for some measurements any day of the week. The same holds for just about any complicated system, not just nuclear power plants.

It has been said that if Statistics, Probability and Metric Space concepts had been better taught to engineers, as ideas instead of recipes, the fact that Fuzzy set theory and Fuzzy logic are unnecessary would have been apparent, and the subject would never have been invented. It is imprudent to buy further into this fight, so I'll stop there, but the reader should know that there *is* a fight. And I daresay by this time the reflective reader will have worked out whose side *I* am on, allowing me to eschew the hypocrisy of pretending to be neutral.

Since writing the above, I have had some discussions with Jim Bezdek, who has a commitment to Fuzziness. Now Jim is an intelligent and thoughtful man, and his views merit serious consideration. Nor is he wholly alone in this.

Bezdek claims that the semantics of Fuzzy Sets and the semantics of Probability theory are quite different. He gives as an example the case where you stagger through the desert, on the point of dying from dehydration, and meet Jim who offers you a choice of two bottles. He describes the one bottle by a `potability' value of 0.95, and the other as a probability of being pure water of 0.95. The latter, he says, represents a gamble: 100 bottles were taken, 95 filled with pure water, five with salt solution in toxic concentration, and a choice made at random. The question is, would you rather take the 0.95 potability bottle, or the other?

Were it me staggering out of the desert, my first response would be to sample the probability bottle, and if water, drink it. My second response would be to threaten to beat Jim over the head until he told me what `potability' means. I know it means `drinkable', but what does the 0.95 mean? Jim might argue that the term is inherently vague and has no answer, but I am bigger than he is and would insist. Does it mean that he took 95 parts of pure water and 5 parts salt, mixed them up and filled the bottle from the mixture? Or maybe it was 95 parts water and 5 parts potassium cyanide. I need to know. More crucially, the sort of data I want is something along the lines of `if you do this sort of thing often, what fraction of people who took the 0.95 potable bottle survived to talk about it? That is, we reduce to counts of possible outcomes that I care about.

The important thing I care about is, will drinking the contents of the bottle shorten my life expectancy or increase it? To be told that it is 0.95 potable is to be told that I have a whacko on my hands who proposes to tease me with meaningless jargon, and I don't take kindly to this even when not dying of thirst. To be told that of the people who drank the same mixture in the past, all complained of mild indigestion but went on walking to the next delicatessen, is to be given some idea of what it means. To be told that 95 percent lived but the other five percent died is again to be given some sort of useful information. But to be confronted with the label and no idea of how to interpret it is merely frustrating. If I found the bottles with their inscrutable labels but no Jim to interrogate, I should sniff the contents carefully and sample them to get some idea of what to do. But I'd do that anyway, since who would believe a label in such a case? What I would do with the (sampled) contents would depend on just how thirsty I was. This is what most people would do, I daresay, and the fact that none of us would bother about the label much tells us something about labels.

This might be thought to miss the point. In making decisions we often take into account things we have been told, and we are often told them with a fair degree of vagueness. We do not know the particular usage of terms employed by the teller, and if a short man tells you to look out for a big bloke at the airport, and he turns out to be of only average height, then one would not perhaps be too surprised that the word `tall' means something different to a short man than it does to a tall one. But we might reasonably suppose that the teller has some reasonably precise definition in his head, even if he doesn't know what it is. We could parade a collection of people of varying heights before our source, and ask him to classify them, and we could conclude that `tall' means, to him, approximately `over five foot nine inches in height'. We don't have the luxury of conducting the experiment, so we have to guess what he means. This is plainly a problem with our ignorance of how someone else uses a term. Probability theory can handle this quite well. We use as data the way we have heard the word `tall' used in the past. This is how we handle much of the vagueness of natural language.

Sometimes vagueness is used to connote indifference, sometimes ignorance, sometimes because only crude approximations are necessary and vague is quick. Few people know their weight to three places of decimals in kilograms or pounds, and few care to. But there seems to be no necessity for asserting that there is some numerical degree of membership in the class of overweight people, and no obvious intepretation of such a number when given.

For these reasons, I cannot take seriously the philosophical foundations of fuzzy sets and fuzzy logic, although the latter makes a harmless exercise in mathematics.

Early in the Nineteenth century, Napoleon, who thought Mathematics important and rather fancied himself as being, spiritually, a Mathematician who had just somehow lapsed into being an Emperor, was talking to Laplace. Napoleon twitted Laplace about the latter's book, *Mecanique Celeste*, which dealt with the origins of the solar system. ``I see that you have made in your work no mention of *le bon Dieu*, monsieur de Laplace". ``Sire", replied Laplace, ``I have no need of that hypothesis".

This is roughly my position on fuzzy logic, fuzzy sets and fuzzy thinking. I can get along better without them.

Next Up Previous Contents

Next: Robots **Up:** Alternative Representations **Previous:** Strings, propositions, predicates and *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Summary of this chapter **Up:** Alternative Representations **Previous:** Fuzzy Thinking

Robots

In the rest of this book, we shall rely rather heavily on the vector space representations of data, because these arise naturally from measurement processes implemented in hardware and usable by robots. They also arise naturally as descriptions of animal nervous systems at the sensory level, where the incident energy or its logarithm tends to get coded in terms of neuronal spike frequencies.

We shall expand further on the details of the methods, metric, neural net and statistical, which have been outlined in the present chapter. We shall be concerned with issues in syntactic pattern recognition arising out of these methods and logically anterior to them, and we shall treat of the dynamical case of trajectories in the representation spaces and their recognition problems. But we shall not, for the reasons given, treat of the AI or fuzzy types of representation.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Exercises **Up:** Basic Concepts **Previous:** Robots

Summary of this chapter

This first chapter has been intended to stimulate some reflective thought on the nature of the problems faced in Pattern Recognition as a prelude to getting into the technicalities. You were invited to put your feet up (except for a short spell in the kitchen), and I hope you did.

The problem of telling men from women by wholly inappropriate measurements of the wrong things is a depressingly accurate paradigm of much pattern classification, and the methods somewhat sketchily outlined are, in the main, those currently in use.

This chapter has surveyed the general ideas used in pattern recognition by looking at very simple examples of the general methodologies. The basic issue of what you measure and how you represent it was discussed in a rather unsatisfactory way, but it was argued that coding objects to be discriminated as points in has more power than alternatives.

Given this choice of coding, pattern classification devolves into the choice of what to measure, which is currently something of a black or at best grey art, and then finding algorithms which can assign a category to a new point. These fall into three major classes, metric, neural net and statistical. Examples of each were sketched, and some rationales for each were discussed. It was pointed out that *unsupervised learning* was a matter of finding clusters in a space and *supervised learning* was a matter of fitting a function from a family of functions to a set of data points on which the function values are known.

The problem of dynamic patterns, that is to say patterns with a temporal ordering on them was mentioned in two cases, the case of trajectories in , as in speech and on-line character recognition, and the case of trajectories in a space of discrete symbols to which the former problems might be reduced.

The problem of more general relationships between points or symbols was mentioned, and disparaging things were said about philosophers, AI practitioners and the Fuzzy Folk. Many promises were made that the ideas outlined rather sketchily would be explained more fully in subsequent chapters.

Various provocative remarks were made, not intended to irritate you, but to get you to state your position.

Next Up Previous Contents

Next: Exercises **Up:** Basic Concepts **Previous:** Robots *Mike Alder*
9/19/1997

Exercises

These are intended to provoke thought. Nothing that makes people think can be all bad.

1. Suppose you are given 1000 digitised images of trees and another 100 images of pin-up pictures of underclad women. It is known that these come from a source of such pictures which has them in that ratio. The images are stored in a computer in some format, and a supply of new images is about to be offered to the computer, each either an underclad woman or a tree, obtained from the same source. You have been consulted by the local Wimmynz Kollektiv to write a program that will delete the image file if and only if it is a picture of an underclad woman. Each image is 512 pixels square and is in full colour. Can you suggest plausible ways of representing the images as points in a suitable vector space so as to make the automatic discrimination of the two classes of image feasible? Which methods of classification would you consider most reasonable and why?

   (Hint: Counting the number of pink or brown pixels might be a reasonable start. More complicated procedures could be necessary for pictures of trees taken in Autumn.)

2. The points $\begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} -2 \\ -2 \\ -2 \end{pmatrix}$ are the good guys. The points

$$\begin{pmatrix} 2 \\ 2 \\ -2 \end{pmatrix}, \begin{pmatrix} -2 \\ -2 \\ 2 \end{pmatrix}$$

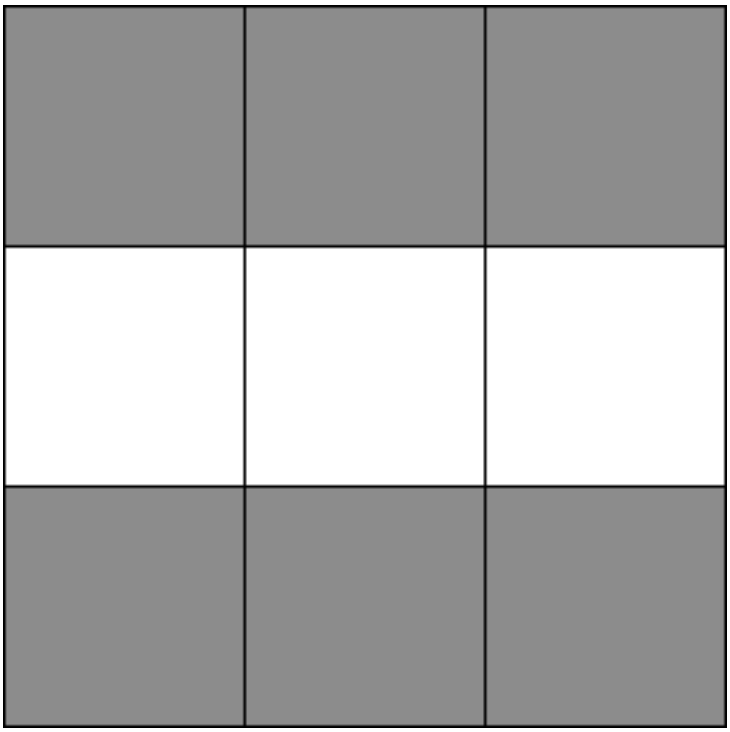
   are the bad guys. Is there a perceptron neural net such as *Fig.1.6* which can discriminate the two kinds of guys?

3. On a 3 by 3 array of pixels, we can draw horizontal bars consisting of white pixels (1) against a black (0) background by writing out the result of a raster scan of bits: thus the image of *Fig.1.12* is coded as (0 0 0 1 1 1 0 0 0) and a vertical bar likewise might be represented as (0 1 0 0 1 0 0 1 0).

   Each of the three horizontal bars and the three vertical bars thus gets coded into a point in $\mathbb{R}^9$. First convince yourself that a single unit neural net cannot distinguish vertical from horizontal bars. Now take three units with weights given by: (1 1 1 0 0 0 0 0 0), (0 0 0 1 1 1 0 0 0) and (0

0 0 0 0 0 1 1 1 0). Let the output of these units go into a `third layer' unit. What weights on the last unit will ensure that its output correctly tells horizontal from vertical bars? (Confirm that it cannot be done by *any* choice of weights.) If you are allowed two more units in the second layer, what choice of weights on these units and the third layer unit will ensure that horizontal can be told from vertical? Can any choice of weights on three units in the second layer guarantee a choice of weights on a single third layer unit which will do the job? Any suggestions?

Figure 1.13: 3 x 3 array of pixels with horizontal bar in middle.



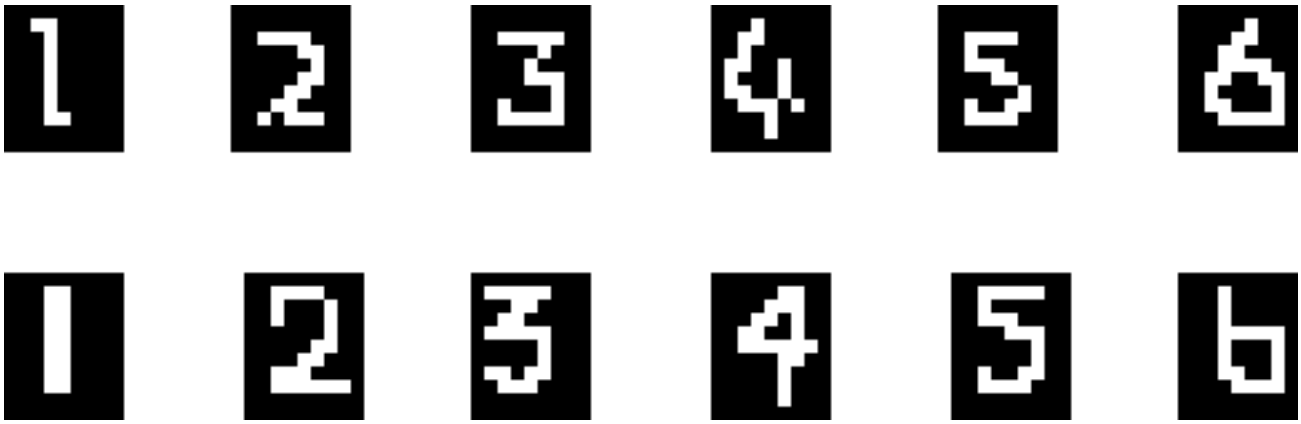
4.

Someone gives you a set of about two hundred 11 by 9 pixel arrays on which have been drawn digits from 0 to 9, as in *Fig1.14*. There are several different examples of, say , a `1', in different locations in the grid, and so there are ten clusters each of twenty points in $\mathbb{R}^{99}$, if you simply code the pixel arrays by doing a raster scan. There has to be a way of coding the points in a lower dimensional space. Find one. Find another. Analyse the advantages and disadvantages of these methods.

5.

Use a scanner on *Fig.1.14* to get the image stored as a TIF file, then use a TIF reader program supplied to enrolled students to display it as an image on a PC or UNIX workstation. It is now stored as an array in the program. Write your own C program which includes the TIF reader as a procedure to obtain the image and use it to test your solutions to the last problem. If you cannot get access to a scanner, email your tutor for an ftp site with the data already scanned for you.

Figure 1.14: Digitised 11 by 9 images of digits



Next Up Previous Contents

Next: Bibliography **Up:** Basic Concepts **Previous:** Summary of this chapter *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Image Measurements **Up:** Basic Concepts **Previous:** Exercises

Bibliography

The following works may be helpful or entertaining. They are selected with the intention of being accessible rather than comprehensive. Duda and Hart's book is one of the original and still one of the best books, although showing some signs of its age.

1. Richard Duda & Peter Hart *Pattern classification and scene analysis* New York: Wiley 1973.
2. Mike James *Pattern Recognition*, BSP Professional Books, Oxford. 1987
3. Frank Rosenblatt, *Principles of neurodynamics : perceptrons and the theory of brain mechanisms.* Washington : Spartan Books, 1962.
4. Nils J. Nilsson *Learning machines : Foundations of trainable pattern-classifying systems* New York : McGraw-Hill, 1965
5. F.N. Stein *Brain Machines* **ANALOG**, May 1975.
6. Don Hush & Bill Horne *Progress in Supervised Neural Networks* IEEE Signal Processing Magazine Vol 10 No.1 Jan 1993 pp 8-39.
7. R.P.Lippmann *An Introduction to Computing with Neural Nets* IEEE Acoustics,Speech and Signal Processing Magazine, 4(2):4-22, April 1987.
8. Jane Austen *Mansfield Park* London: The Zodiac Press, 1957

For alternative views on Fuzzy Sets, see

1. Walter J.M. Kickert *Fuzzy theories on decision-making : a critical review* Leiden : M. Nijhoff Social Sciences Division, 1978 (Frontiers in systems research ; v.3)
2. James C. Bezdek, *Pattern recognition with fuzzy objective function algorithms* New York : Plenum Press, 1981. (Advanced applications in pattern recognition)
3.

Bart Kosko *Neural networks and fuzzy systems : a dynamical systems approach to machine intelligence* Englewood Cliffs, N.J.: Prentice-Hall, 1991, c1992

The book by Bezdek is unlikely to diappoint, and may be recommended to those who do not accept the views of the present writer or at least wish to scrutinise them critically. Note that the last book cited contains in its title almost every power word that has been trendy in the last decade. It needed only genetic algorithms, chaos and feminist thought to be guaranteed a place on every bookshelf.

And for the standard opinions of the Artificial Intelligentsia, the first somewhat out of date but well worth reading (partly because of that) see:

1. Herbert A. Simon *Sciences of the Artificial*, MIT Press, Cambridge, Mass. 1982
2. Patrick Henry Winston, Artificial Intelligence Artificial intelligence 3rd ed. Reading, Mass : Addison-Wesley, 1989

For a somewhat less standard approach to AI with a good deal of charm, see:

1. Douglas R Hofstadter *Gödel, Escher, Bach, An Eternal Golden Braid*, Basic Books, 1979
2. Douglas R. Hofstadter *Metamagical Themas: Questing for the Essence of Mind and Pattern* Bantam New Age Books 1986.

The exponents of AI can frequently write well, as in the above cases, and have sometimes put in a great deal of thought. The objection of a Mathematician is that the thought is, in general, amiably amateurish. This sour remark may reflect disenchantment or hindsight or both.

Doug collected a Pulitzer prize for GEB, which is a certain kind of warning. Still, anybody who likes the Alice books can't be all bad. These are delightful books to read while slightly drunk on a good champagne.

Next Up Previous Contents

Next: Image Measurements **Up:** Basic Concepts **Previous:** Exercises *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Preliminaries **Up:** An Introduction to Pattern **Previous:** Bibliography

Image Measurements

The chapter you have just read was a general survey of the ideas and issues in Pattern Recognition, and may have left you feeling a little insecure. After all, this is a Mathematics course in those parts of mathematics useful for Information Technology, and there was hardly any mathematics in the first chapter. Well, actually there was some mathematics, but it was said in English instead of algebra, and you may not have recognised it.

From now on we shall be dealing with more concrete objects and for the present we shall concentrate on image recognition. This is part of the plan to say something about how robots can be made to see the world.

In this chapter we shall consider various kinds of images such as might be produced by a camera or scanner, and the methods which may be employed to code them as points in so as to accomplish recognition of the objects in the images. The underlying assumption is that a robot is looking at some collection of objects and needs to make decisions based on a pre-defined human classification of the types. The robot has to analyse a set of pixels in order to make a classification. In the next chapters I shall discuss statistical and ANN methods of recognition of the resulting vectors, but for now I attend to the matter of how to extract vectors from static images. Some of the algorithms will be described algebraically, reassuring you if you feel that the medium is the message, but some will be given in English. Simple algorithms can be taken from English into C without passing through algebra on the way, and it is often easier to understand why they work.

Concntrating on images obtained from a video-camera or scanner may strike you as a bit limited, since there are other sources of information than cameras. On the other hand, the general problems are best approached through concrete instances, and much of the conceptual apparatus required for thinking about measuring images is capable of extension to thinking about measuring other things.

The process of performing an operation on an image to obtain a vector of numbers is often referred to as *feature extraction*. Each number in the vector is sometimes said to describe a `feature'; this somewhat mystical terminology might leave you wondering what a `feature' actually is, and the answer would appear to be that anything you can do to an image to get a number out is detecting a `feature'. In various parts of the literature, a `feature' may mean a dark pixel, a bright pixel, a lot of bright (or dark) pixels, an edge, or some combination of the above, as well as other quite different things. Some of this confusion will be resolved in the chapter on syntactic pattern recognition.

Because I don't know what a feature is and therefore feel shy about using the term, I shall simply talk about making measurements on an image (or indeed any other objects), and a suite of measurements on such an object produces a sequence of numbers. A *measurement* of an object is any operation performed on the object which yields a real number.

An integer or boolean value is treated as a particular case of a real number for these purposes, but we shall have a preference for real real numbers, expressed, of course, to some finite number of decimal places.

For most of this chapter, I shall deal with *binary* images, that is to say images where the pixel is either black (0) or white (1) because the recognition problems are sharpest here. Colour images or monochrome greyscale images are, of course, more common, but the recognition problems become overshadowed by the image processing problems- which belong in a different book. In the later sections I shall discuss the complexities introduced by having greyscale and colour images to recognise.

Since there are an awful lot of different things which can be found in an image, even a binary image, we approach the subject by stages, dealing with the complications progressively. I shall focus on a particular problem so as to concentrate the mind; I shall investigate the problem of *Optical Character Recognition*, so as to sharpen the issues to concrete cases. Virtually everything that applies to recognising characters applies to other types of binary image object, except that the other objects are usually harder to deal with. It might be useful to contemplate Chinese or Arabic characters occasionally, or a handful of nuts and bolts in silhouette, so as to avoid parochialism.

Quite a lot of what I shall describe here is treated in books on Image Processing, and parts of what is done on books on Machine Vision or Computer Vision, and there is some intersection with the material in books on Computer Vision.

This is *not* a book on Image Processing, but it will be necessary to outline the ideas and relevance of Image Processing material. My treatment will therefore be sketchy, and the bibliography should be used to fill out the details. Since many books have been devoted to what is covered in this chapter, you may take it that the treatment will be lacking in that depth which the intelligent reader quite properly craves; in exchange you have here the opportunity to get an overview of what the stuff can be used for. An honest attempt to do the Exercises at the end of the chapter, if necessary with the aid of some of the books referred to in the bibliography, will provide a crash introductory course in Image Processing. This is not a substitute for a Computer Vision course, but a complement to it.

The first chapter gave an overview from such an exalted height that the reader may have been left feeling a touch of vertigo. It is now necessary to take one's feet down, pour the drink into the cat's bowl, lace up one's running shoes and sober up. A certain amount of the nitty-gritty is waiting further down the alley with a sand-filled sock.

- Preliminaries
  - Image File Formats
- Generalities
- Image segmentation: finding the objects
  - Mathematical Morphology
  - Little Boxes

    - Border Tracing
    - Conclusions on Segmentation
- Measurement Principles
    - Issues and methods
    - Invariance in practice
- Measurement practice
    - Quick and Dumb
    - Scanline intersections and weights
    - Moments
    - Zernike moments and the FFT
        - Historical Note
    - Masks and templates
    - Invariants
    - Simplifications and Complications
- Syntactic Methods
- Summary of OCR Measurement Methods
- Other Kinds of Binary Image
- Greyscale images of characters
    - Segmentation: Edge Detection
- Greyscale Images in general
    - Segmentation
    - Measuring Greyscale Images
    - Quantisation
    - Textures
- Colour Images
    - Generalities
    - Quantisation
    - Edge detection
    - Markov Random Fields
    - Measurements
- Spot counting
- IR and acoustic Images
- Quasi-Images

- Dynamic Images
- Summary of Chapter Two
- Exercises
- Bibliography

Next Up Previous Contents

Next: Preliminaries **Up:** An Introduction to Pattern **Previous:** Bibliography *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Image File Formats **Up:** Image Measurements **Previous:** Image Measurements

Preliminaries

This section discusses the low level technicaities of getting images stored in a computer and in particular the competing formats.

- Image File Formats

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Generalities **Up:** Preliminaries **Previous:** Preliminaries

Image File Formats

There are currently many different ways in which images are regularly stored inside a computer, and for each of them there has to be a way of reading the image into some sort of viewing program to actually *look* at it, and the pixels have to be stored in an array in memory in order to process the image or subimage. If we wish to restrict ourselves to binary images, then a single bit can be used to store the value of a pixel at a particular location. A binary file can therefore be stored as a long list of bits, but it also needs some information to say how many there are and what the dimensions of the array are.

The graphics systems of different computers also vary a lot. This presents the writer of a text book on image processing or pattern recognition with certain problems. What machine does one suppose the reader has access to? What programming languages does one assume he can use, and what make of scanner or camera is he going to be able to get his hands on?

These problems are similar to the problems faced by a web based system too. In designing this course we had to make choices about what kind of hardware you would have, and what kind of operating system it would be running. The decision was to go with the most common systems used in industry as far as the hardware was concerned, and to discuss the software that could run on these systems. So we are going with the Intel Pentium based machines despite the many virtues of the alternatives.

It is possible to get around these matters by writing at a level of abstraction so high that the reader will not actually connect up what he reads with actual images or patterns, but I find this unappealing. I have agonised over these issues and concluded that there is not much use tackling serious pattern recognition without a UNIX workstation, and that X-Windows is currently the *de facto* standard for graphics. At the same time, PC's are widely available and many graphics acquisition cards are available for these machines at low price. Rather than choose between these competing alternatives, I have opted for both of them . I shall start off by supposing that the scanner is attached to a PC and that images can be stored in the common *.tif* files.

TIFF, short for Tag Image File Format, is a convention for storing a variety of images, and since it is a common one and most scanners can read and write such files, I shall refer only to such files. Other standards such as GIF, JPG, MPG and the page description languages can generally be converted, for a particular image, and I do not want to have to write programs for them all.

One of the many programs available with this splendid book, at no extra cost to enrolled students, is a TIF reader, which will take a small binary TIF file and display it on the screen of a PC. It stores the image internally in an array, which means that the enthusiast can monkey about with it to his hearts content. At a later stage in the book, the reader will be introduced to a somewhat beefier X-Windows variant of this program for reading larger images in many colours into a workstation. The reader should certainly acquire the art of scanning documents; it is easy and gives a satisfying sense of power for very low expenditure of effort.

Unless you live far from the madding crowd, there are almost certainly print shops near you which will do scanning, writing the tif file onto a floppy disk. If you *do* live far from the madding crowd, and don't already own one, scanners may be purchased at reasonable cost

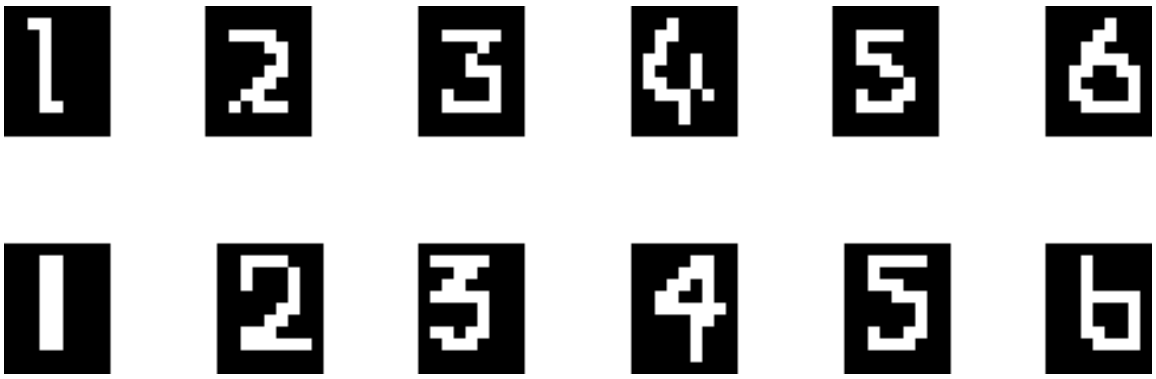
I shall suppose then that the images that follow can be obtained as *.tif* files output from a scanner, and that the transformation to an array of pixels is accomplished.

Next Up Previous Contents

Next: Generalities **Up:** Preliminaries **Previous:** Preliminaries *Mike Alder*
9/19/1997

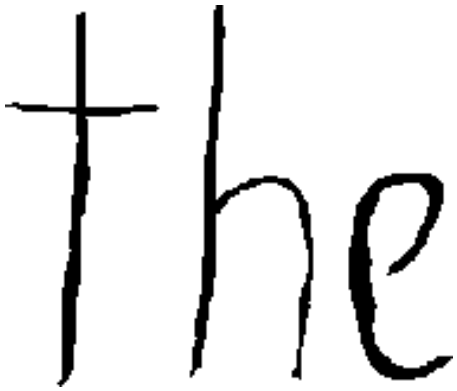
Generalities

Let us start then by supposing the simplest sort of image, a *binary* array of pixels, say 256 by 256. Each pixel is either black (0) or white (1). Hardly any modern equipment actually produces such images unless you count photocopiers, and I have doubts about them. Video cameras and scanners usually produce larger arrays in many colours or grey levels. Monochrome (grey scale) cameras are getting hard to find, and scanners will be following this progression soon. So the images I shall start with are rather simple. The image in *Fig.1.14* , reproduced here for your convenience, is an example of the kind of thing we might hope to meet.



Getting an image to this simple, binary state may take a lot of work. In some forms of image recognition, particularly in *Optical Character Recognition (OCR)* the first operation performed upon the image is called *thresholding* and what it does is to take a monochrome image and rewrite all the light grey (above threshold) pixels to white, and all the dark grey (below threshold) pixels to black. How to make a judicious choice of threshold is dealt with in image processing courses, and will not be considered here. Although a little unphysical, because real life comes with grey levels, the binary image is not without practical importance.

Figure 2.1: A handwritten word.

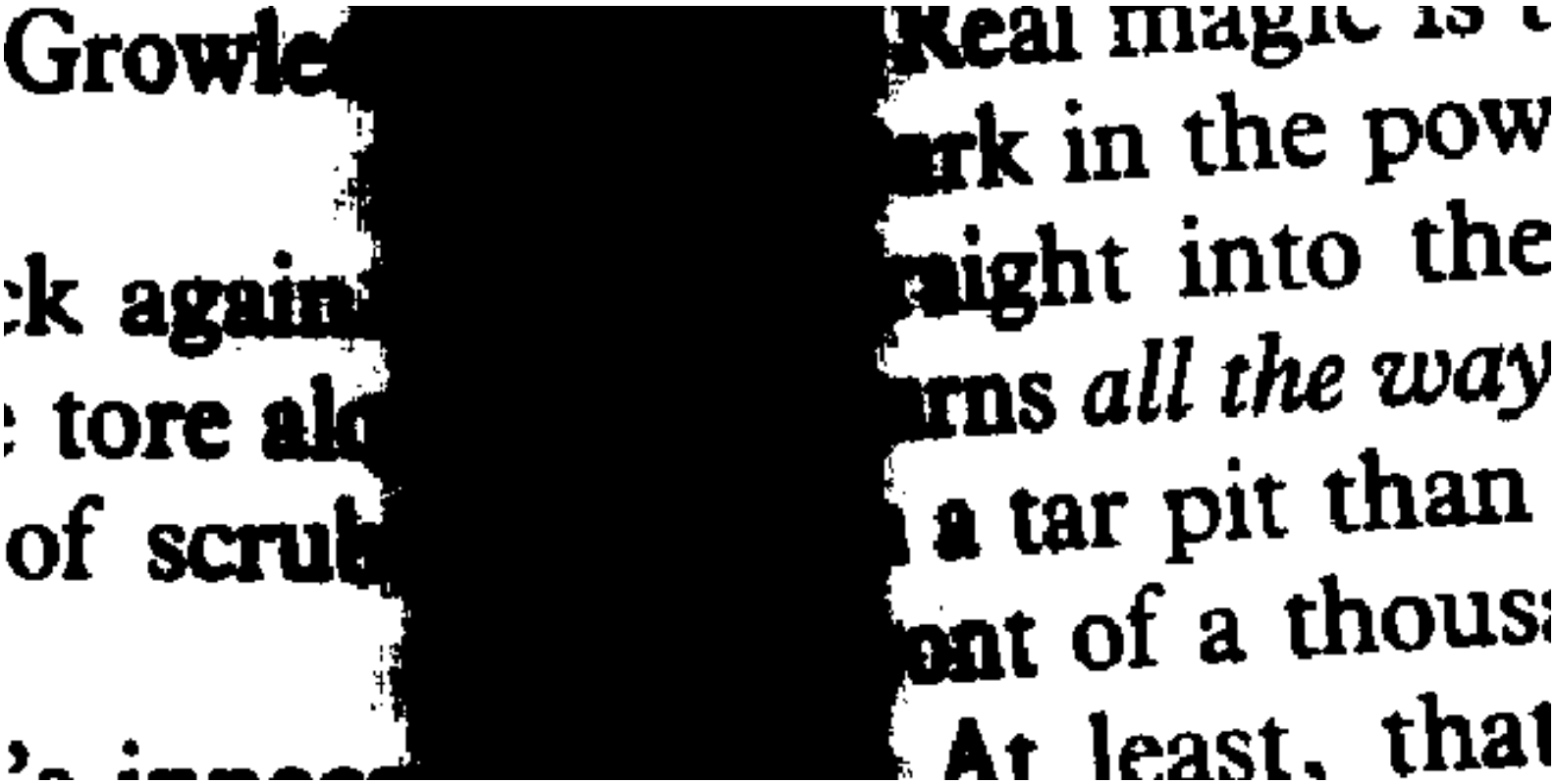


The result of digitising and thresholding a piece of handwriting is not unlike *Fig.2.1.*, where the word /the/ was hand drawn by mouse using the unix *bitmap* utility. It is reasonable to want to be able to read the characters, /t/, /h/, /e/ by machine. For handwriting of the quality shown, this is not too difficult. It is, however, far more difficult than reading

printed script, and so we shall treat the printed case first.

In order to make the problem moderately realistic, we look at the output of a scanner, or at least a bit of such output, when applied to two pages of text taken from a book. (*Fig.2.2.*). The quality of the image is not, we regret to say, up to the quality of the book. Such images are, however, not uncommon, and it is worth noting a number of features.

Figure 2.2: A sample of scanned text from a book.



- The most manifest oddity is the black band down the centre between the pages, where the scanner had trouble with the curvature of the book. The result is that (1) the leftmost characters of several words are unreadable even by human beings and (2) there is some distortion of those characters which *are* readable.
- There is noise consisting of black pixels scattered around the boundary of the central black band and intruding onto the characters.
- The quantisation at 72 dots per inch is sufficient to ensure that no two characters are actually identical. A comparison of the letters /h/ or /o/ for example in their several occurrences shows significant differences in the actual array of pixels.
- Italicisation of part of the text means that we are not, as might be hoped, dealing with a single font.
- The lines of text are not horizontal, the left page and the right present different angles for what, in an ideal world, would be horizontal lines of text.
- Our first problem, before trying to recognise a character, is isolating it from the other characters and the noise, and since we want to preserve the order of the characters when we translate from pixels into ASCII, we need to find lines and order *them* top down, then we need to scan a line from left to right and isolate the characters, including the white spaces between the `objects', since a translationthatlookedlikethis would be, rightly, thought objectionable.

The problem of reading such text is therefore far from trivial, even if we stipulate somewhat higher quality

reproduction than is present in *Fig.2.2*. In doing some serious reading of a document such as a newspaper, large and uncomfortable problems of working out which bits of text belong together, which bits of an image *are* text and which bits are the pin-up girl, and similar high level segmentation issues arise. Any illusion that the problem of reading text automatically is easy should be abandoned now.

Next Up Previous Contents

Next: Image segmentation: finding the **Up:** Image Measurements **Previous:** Image File Formats *Mike Alder*
9/19/1997

Image segmentation: finding the objects

How can we tackle the problem of writing a program which will read the text of *Fig.2.2.*? When I say `read the text', I mean of course that the input will be an image file, such as that from which *Fig.2.2.* was extracted, and the output will be a sequence of ASCII characters, possibly with some additional formatting information to say where the characters are on the page. This is the inverse function to a printer, which takes an ascii file and turns it into black marks on white paper; together with a scanner, a text reading system inverts that.

This is a clear case of a Pattern Recognition problem of an eminently practical type. Typically image files are very big, and text files which contain the data from which an image of text was obtained by printing are very much smaller. Also the latter can be read into your favourite word processor and the spelling checked, something which cannot be done with an array of pixels.

There are clearly pre-processing problems which must be gone through in order to identify lines of text in the image, and to work out where the characters actually *are*. These are not negligible and something must be said about them.

Figure 2.3: Postcode digits 6512

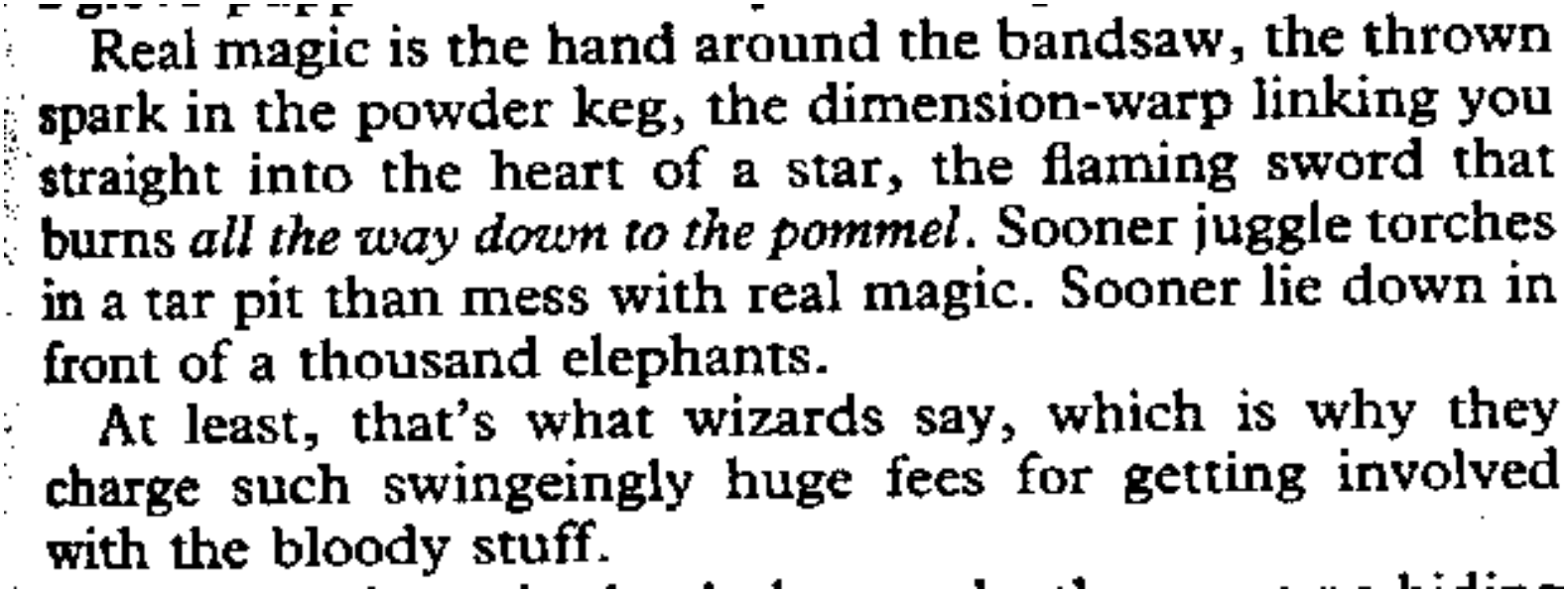
The general problem of image segmentation, chopping an array of pixels up into different bits, is fraught with difficulty: It is easy to run two letters together in an image so that they overlap, and for the eye to still be able to read them as two separate things. A program which kindly lumped the two together as one, would not make automatic recognition any easier. The proposition that the objects are going to come neatly separated by white spaces around them is simply not true. The postcode in *Fig.2.3* is handwritten, it is easy to read, but segmenting it automatically is best done in hindsight once you have worked out what the characters are!

By choosing to deal with printed text of sufficiently high quality we can make our lives much simpler, but there has to be something intrinsically unsatisfactory about a method which cannot generalise to cases which human beings find easy.

With the cautionary note out of the way, and bearing in mind for later chapters that there has to be a better method, let us proceed to examine the standard means of proceeding with the image of *Fig.2.4*, which is a

somewhat cleaner and larger version of *Fig.2.2.* Anyone who can identify the source of the text is entitled to a prize, but already has it.

Figure 2.4: A bigger and better sample from the same book.


Real magic is the hand around the bandsaw, the thrown spark in the powder keg, the dimension-warp linking you straight into the heart of a star, the flaming sword that burns *all the way down to the pommel*. Sooner juggle torches in a tar pit than mess with real magic. Sooner lie down in front of a thousand elephants.

At least, that's what wizards say, which is why they charge such swingeingly huge fees for getting involved with the bloody stuff.

First we find the lines of text. Moving a long way from *Fig.2.4* and squinting at it with eyes half closed, one sees more or less horizontal bands where the words blur together; these are our lines of text. One way of finding them is to `fuzz' the image by what are known as *Mathematical Morphology* techniques. The ideas of Mathematical Morphology are very simple (and usually made difficult by being expressed in formalism, a favourite method by which esoterrorists try to impress the innocent). We proceed to elucidate the easiest case; references to more comprehensive accounts may be found in the bibliography at the chapter's end.

- Mathematical Morphology
- Little Boxes
- Border Tracing
- Conclusions on Segmentation

Next Up Previous Contents

Next: Mathematical Morphology **Up:** Image Measurements **Previous:** Generalities *Mike Alder 9/19/1997*

Mathematical Morphology

Let A and B be subsets of $\mathbb{R}^2$. Then the *dilation* of A by B is the set C defined by

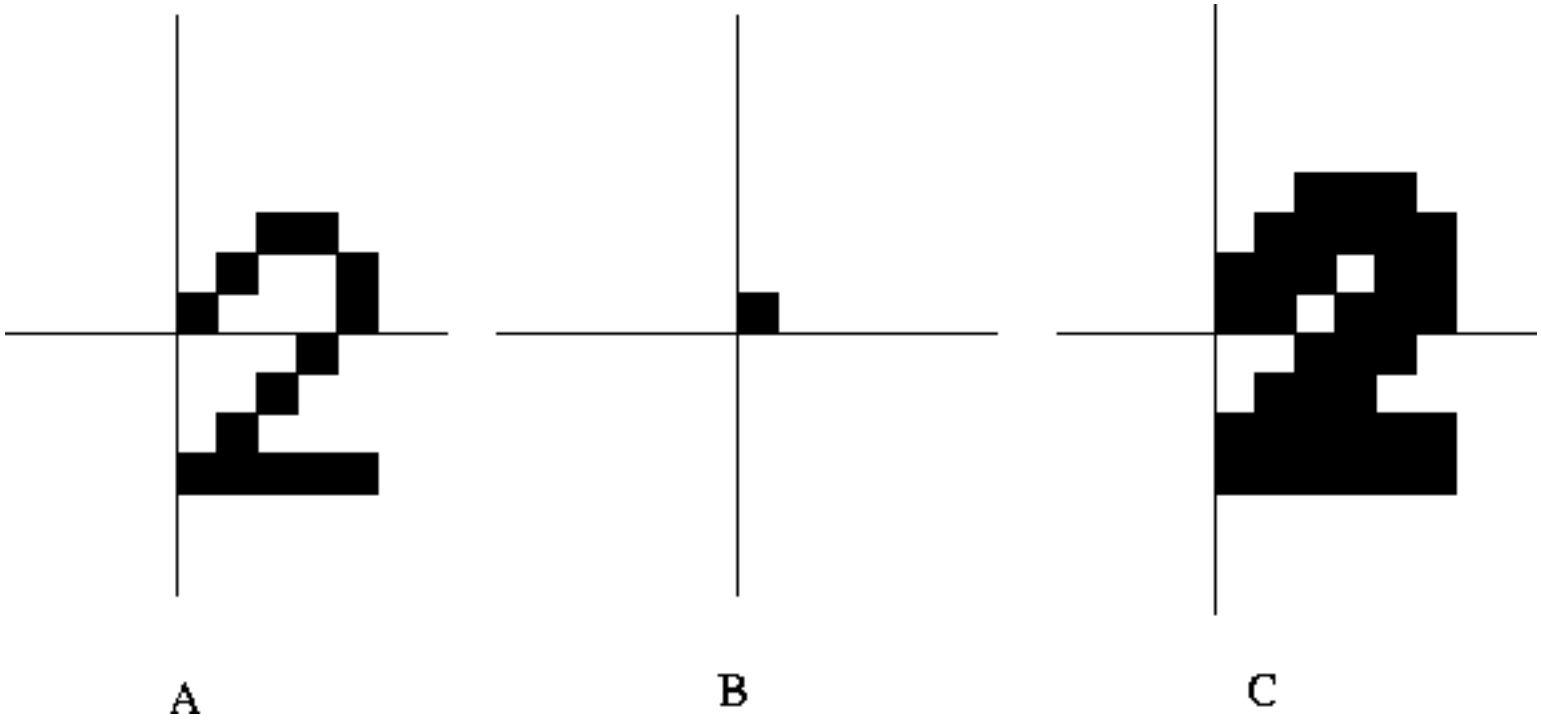
$$C = A \oplus B = \{\mathbf{x} \in \mathbb{R}^2 : \exists \mathbf{a} \in A, \exists \mathbf{b} \in B, \mathbf{x} = \mathbf{a} + \mathbf{b}\}$$

where the addition is, of course, the vector addition in that well known vector space $\mathbb{R}^2$.

In applications, A and B will be finite sets of pixels. We obviously have that $A \oplus B = B \oplus A$ but we shall tend to choose a particular B and think of it as operating on a lot of different A's. In this case, B is referred to as the *structuring element*.

Example: If A and B are the sets shown in *Fig.2.5*, where the origin and axes are as marked, we see that $A \oplus B$ is the `thickened' version shown as C. By making B bigger and approximately disk shaped, we can fatten A quite out of recognition. In this example, each black square is a pixel, and if we think of the corners of the square which is B being at the points $\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, every point **x** of A is expanded to a copy of the unit square stuck onto **x** at its lower left corner. So A is `blobbified' by B; equally, B has been blobbified by A.

Figure 2.5: $A \oplus B = C$



For digital images, the dilation procedure is somewhat simpler; we have the zero which is usually a part of the structuring element, and some points at integer co-ordinates. We simply move the structuring element *B* and place the zero at a pixel of *A*, and then fill in all the other pixels of *B*. Then we repeat for each of the pixels of the original *A*.

Similarly, it is possible to have an *erosion* operation, $A \ominus B$ which treats *B* as a mask with holes in it where the pixels are. Then to get $A \ominus B$, place the origin of *B* at a pixel of *A*, and look to see which of the pixels of *A* can be seen through the holes in *B*. If there are any holes in *B* without points of *A* visible through them, mark the pixel of *A* visible through the origin of *B* for deletion. Repeat for each pixel of *A*, and finally remove all the pixels marked for deletion. Formally:

$$C = A \ominus B = \{\mathbf{x} \in \mathbb{R}^2 : \forall \mathbf{b} \in B, \mathbf{x} + \mathbf{b} \in A\}$$

Note that I have assumed in my explanation of how erosion is carried out that the origin is in *B*.

Exercise: What difference does it make if it isn't?

Repeated application of erosion by a suitable structuring element B can be useful for *thinning* sets. Carried too far, sets can easily be eliminated altogether.

Exercise: Draw a grid of lines, label a horizontal line in the middle somewhere as the X-axis, a suitable vertical line as the Y-axis, and draw pixel dots on the intersections at enough points to pick out a big fat character like the blobbified /2/ of *Fig.2.5*. This is A. Now take B to be the origin and its 8 nearest neighbours, on a separate grid drawn on transparent film. Do the dilation operation, then the erosion operation on the result. Start again and do it in the opposite order. Do the two operations commute? If the

origin is omitted from B, and we use the formal definition, what effect does this have?

Remark There are hardware implementations of many such operations which are extremely fast. See Wojciech Kuczborski's Ph.D. thesis, coming real soon to a store near you, and referenced in the bibliography at this chapter's end. Well, you can write to Wojciech care of The Centre for Intelligent Information Processing Systems (CIIPS) at the University of Western Australia and ask nicely for a copy.

If we blobbify (or dilate) the set of pixels in the image *Fig.2.4* by a horizontal bar of pixels, we `stretch' pixels horizontally. This will join up adjacent letters and blur them in a horizontal direction without having any effect vertically. It is childishly simple to write a program which goes over an array of pixels and dilates them horizontally: every time a black pixel is found, the new image has, say, black pixels inserted to the left and right of the location of the given pixel, as well as copying the original pixel. I assume, of course, that we are dealing with black images on a white ground.

Doing this to *Fig.2.4*, if necessary several times on the result of doing it previously, gives something close to horizontal bands. The page has its text at an angle, so the bands are not very horizontal, but it is now easy to find the spaces between the lines and also to determine the height of the bands that used to be characters. The lines of text can be numbered consecutively from the top and their direction specified.

Of course, there are ways of finding the lines of text which do not require us to apply morphology methods, but the methods will have other uses, and this seemed like a good place to introduce them.

I said it was easy to find the lines, and it is, if you don't mind doing the sort of ghastly hack that occurs naturally to the more depraved kind of programmer. If your aesthetic sensibilities are revolted by this sort of thing, good. There are better ways and we shall come to them in due course.

Next Up Previous Contents

Next: Little Boxes **Up:** Image segmentation: finding the **Previous:** Image segmentation: finding the

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Border Tracing **Up:** Image segmentation: finding the **Previous:** Mathematical Morphology

Little Boxes

Next we can start at the left hand end of each line (of the original image, not the dilated one!) and locate the first black pixel. This may be the bottom half of a character of text in the line above, the page number of the page, some noise from the book spine via an out of focus scanner, or a bit of a thumb-print or soup stain or worse. Since we have estimates of the character height obtained from the thickening, we can make an estimate of where the bottom of the character should be if we actually have the top left hand pixel of a character, and also have some ideas about where the rest of it should be. These will be very crude, because we might have an /o/, or a /b/ or a /p/ or a /q/, not to mention capitals or punctuation. If we have confidence in our knowledge of the font, we can look for vertical (relative to the direction of the line as horizontal) and horizontal separating lines between which our character may be found.

It is common practice in the case of the Roman alphabet to fit slightly slanted lines from South-South-West to North-North East because, as a careful examination of the italic text shows, there is not in general a vertical separating line between well spaced characters in italic fonts. This can also happen with well spaced non-italic fonts as when TEX places the characters in a word such as

WAVE

and it should be noted that there may be no line slanted like: `/' separating characters, either.

It should be plain that separating out characters from each other and putting boxes around them is not usually the simple procedure one might have hoped for.

Next Up Previous Contents

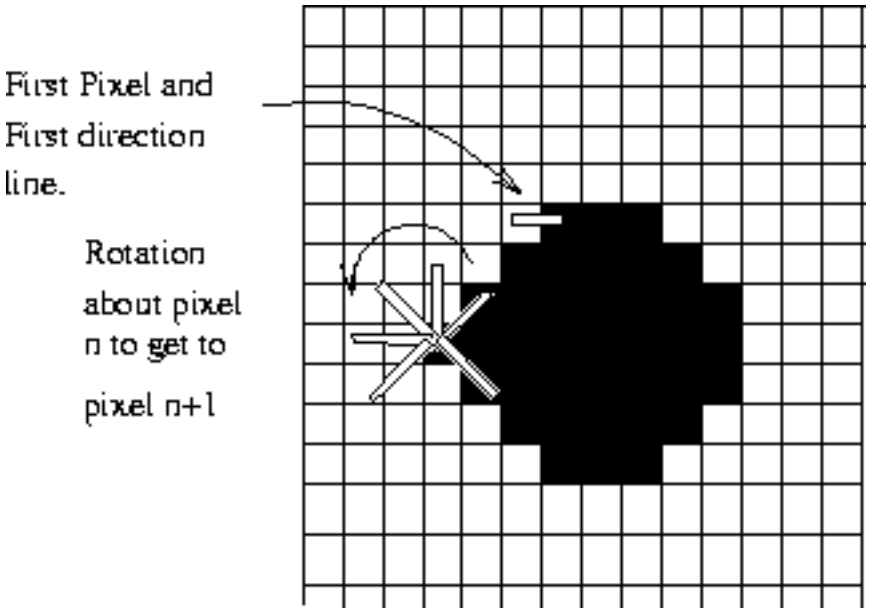
Next: Border Tracing **Up:** Image segmentation: finding the **Previous:** Mathematical Morphology *Mike Alder*
9/19/1997

Border Tracing

A more cautious approach than trying to find a box around each character, is to trace the exterior boundary of the black pixels. Then we can work out from the length of the boundary something about the size and shape of the object we have found. This should distinguish noise from characters, given reasonable luck and a good quality image. It also puts a box around the object, although rather an irregularly shaped one. Still, better a weird shaped box than no box at all. At least, we can segment the image into objects.

As we have noted, in the case of handwritten characters it could be a mistake to imagine that the segments we get this way are going to be characters. They could be bits of characters, or more often several characters joined together. For good quality printed text however, border tracing stands a reasonable chance of isolating characters. We therefore discuss the problem of finding the border pixels of a connected block of pixels. We then regard the character as the set of pixels lying inside this rather irregular `box'.

Figure 2.6: Border Tracing



The algorithm described briefly here (and which may be found, with some additional nuances, in some of the references below, particularly Haig, Attikiouzel and Alder) is one which the most innocent would devise unaided.

First we assume that the image has a white border around it, and we have to find the bounding pixels of clumps of black pixels. We therefore start at the top left of the image at a white pixel, and scan horizontally until we encounter the end of the row or a black pixel. In the former case we do a line feed carriage return to start at the left side of the next row down. If we are at the bottom row already and there is no next row, we have scanned the page and stop.

If we have found a black pixel, we draw an imaginary line from the pixel just to the left of it to the black pixel. We note the direction of this line, and the co-ordinates of the black pixel. Then we rotate the line, about the black pixel, in the positive direction (anticlockwise), until it meets another black pixel adjacent to the first black pixel, or returns to its original position. In the latter case the black pixel is isolated; we mark its location, delete it and start again. If we find another black pixel then it is in one of the eight positions adjacent to the first black pixel. Call the black pixel we are currently standing on pixel n. When $n = 1$ we are at the first black pixel, but this works for all of them. We now move to pixel $n+1$, note its location, note the direction from pixel n to pixel $n+1$, and take the line joining pixels n to $n+1$ and rotate it, about pixel $n+1$, in the positive direction, until it finds an adjacent black pixel which becomes pixel $n+2$. This process is repeated until pixel $n+1$ is a previously listed pixel and the line from pixel n to pixel $n+1$ is the same previously listed direction. This last condition is very necessary, as it is easy to re-enter a pixel from different directions without circumnavigating the set. The reader should examine carefully the border tracing program and try it out on some sets to see if it locates borders in a sensible way.

Once a black object has been found, the pixel set of its border is determined by this algorithm, and this set of pixels *together with every pixel which is inside this border* is stored as an object and then deleted from the page, and the scan starts again. Only when the page has been blanked does the procedure terminate.

Deleting the pixels inside the border sounds easy but may not be if the border is not convex. To see if a pixel is inside the border, draw a half ray starting at the pixel and extend to the boundary. Count the number of times this intersects the border. If the number is even, the starting pixel is outside, otherwise it is inside or on the border. This works except in some `bad luck' cases where the half ray hits an extremal point of the boundary. So wobble the line and the starting pixel a little bit and try again. If a pixel is inside the border and not on it, every adjacent pixel is either inside or on the border. A `floodfill' operation stopping at the border and starting inside it can be used to label the points to be removed. Of course, a border may have no interior points.

There are more efficient ways, but it is relatively easy to prove rigorously that this method will work.

Well, sort of. If the image has a black border, or a black box around a character, as in , and if the border is broken, then the algorithm will fail to find the desired set.

Similarly, it will separate out the components of colons, semi-colons and characters such as i and j which are disconnected.

Noise, black lines down the middle where the centre of the book got miscopied, smudges and the like will be picked out and will have anomalous shapes: the single pixel dots can be eliminated at this stage. Knowing the approximate size of a character can help remove the odd spots, although it is easy to get rid of the punctuation as well.

If the input text comes in a wide variety of fonts, as for example the page of a newspaper or magazine, or is mixed with graphic images, then the complications of segmentation can be formidable and go beyond

the scope of this book, and, indeed, most others.

Next Up Previous Contents

Next: Conclusions on Segmentation **Up:** Image segmentation: finding the **Previous:** Little Boxes *Mike Alder*
9/19/1997

Conclusions on Segmentation

The exercises at the end of the chapter will give you some practice at doing some of this pre-processing. Later we shall discuss less flagrantly *ad hoc* methods of tackling the problem.

Separating out characters by putting little boxes on them, little rectangular boxes or little parallelogram boxes, little rhomboid boxes, or even little irregular boxes obtained by border following, allows us to start thinking about how to code each character as a point in for some *n*.

Contemplating a page of medium bad handwriting or *Fig. 2.3* will suggest that this is not the way people do it. We can handle the case of touching characters or broken characters with no trouble at all, and touching or broken characters are far from uncommon, even with reasonable quality printing. So there is something more fundamental to be sorted out here.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Issues and methods **Up:** Image Measurements **Previous:** Conclusions on Segmentation

Measurement Principles

- Issues and methods
- Invariance in practice

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Invariance in practice **Up:** Measurement Principles **Previous:** Measurement Principles

Issues and methods

Let us now suppose that we have isolated a character by some acceptable process, possibly only tentatively, and that we now wish to proceed to the recognition/classification stage. The intervening step is to measure some properties or characteristics or *features* of the character, so we will be able to compare and contrast the results of doing the same measurements on other characters. There are several well established methods which will shortly be described, but there are a few issues to be contemplated before going into a list of them.

If I suggest a suite of measurement operations, certain properties may or may not hold for my selection. For example, it is conceivable that my measurements would be quite unaffected by simply shifting the character by one pixel to the right. More generally, I might choose to make measurements that would be the same no matter where the character is on the page. In this case we say that the measurement process is *invariant* under shifts. Similarly, if a measurement process were invariant under scaling, it would yield a set of numbers which would be the same for any two characters which had one simply a scaled up version of the other, and if it were invariant under rotations, it would not matter how much you rotated a character, the measuring process would give the same output. These ideas derive from the continuous case, and have to be modified a bit for the case of pixel arrays, but clearly there is some vague and approximate notion of invariance in the discrete case.

Even if the measuring process were not shift invariant, it would certainly simplify recognition if the resulting vector of numbers did not change beyond recognition if the character were moved one pixel to the left. Since we may very well want to know where the character is, complete shift invariance is undesirable, but it would be useful to have the measurement process partitioned into a part which tells you where the character *is*, another part which tells you how big it is, another part where the numbers tell you if it has been rotated, another part perhaps telling you if it has been transformed in some other standard way, such as a change of font, and finally a component which would be the same no matter how much the character had been scaled or rotated or where it was. It is not too much to hope that some of these properties would be found to hold for a well chosen suite of measurements. Conversely, a suite of measurements which had every number changed completely by shifting the character one pixel sideways is going to be less appealing.

So it is possible to think a little bit about what constitutes a sensible sort of system of measurements rather than to jump on the first such system that occurs to you. This can save a good deal of time in the longer run.

Next Up Previous Contents

Next: Invariance in practice **Up:** Measurement Principles **Previous:** Measurement Principles *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Measurement practice **Up:** Measurement Principles **Previous:** Issues and methods

Invariance in practice

Suppose we have put a box around a putative character. Suppose that the box is of a reasonable size to contain some character, and that the top and bottom edges are parallel and can confidently be declared as horizontal. This is a somewhat courageous commitment because, as inspection of many a photocopy will show, text lines are often curved near the spine of the book, but we proceed in a spirit of optimism.

The first prudent thing to do is to contract the resulting box by making sure the black pixels inside it are touching the edges. A big box with a lot of white space against the right hand edge has more pixels in it than necessary. The next prudent thing to do is to scale it to a standard size and shape. It is prudent because the characters may be deformed and this will normalise them, and also because it is easier to compare like with like. The effect of this on an /o/ will be very different from its effect on an /i/ of course. What constitutes a good size for the standard box will depend to some extent on the font. If the box was a parallelogram which fitted an italicised character, the transformation back may de-italicise it. And it may have rather an odd effect on some characters, making /V/ much to much like /U/ if taken to excess. Some opportunity to experiment with various transformations is given the reader in the exercises.

Next Up Previous Contents

Next: Measurement practice **Up:** Measurement Principles **Previous:** Issues and methods *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Quick and Dumb **Up:** Image Measurements **Previous:** Invariance in practice

Measurement practice

- Quick and Dumb
- Scanline intersections and weights
- Moments
- Zernike moments and the FFT
  - Historical Note
- Masks and templates
- Invariants
- Simplifications and Complications

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Scanline intersections and weights **Up:** Measurement practice **Previous:** Measurement practice

Quick and Dumb

If we normalise into, say, an 11 by 9 array, we can rewrite the characters into standard form. Then we could, if desperate for ideas, take each character as a point in $\mathbb{R}^{99}$. This is not a good idea, although it has been done. The main reason it is not a good idea is because one extra pixel in the wrong place can give vectors which are totally unrelated: a single pixel can shift a vertical line one pixel to the right with no trouble at all. It would be nice if a horizontal shift of a character by one pixel column were to have minimal effect on the placing of the point in $\mathbb{R}^n$. Another reason it is a bad idea, is that the dimension of the space should be kept as small as possible. The reasons for this are subtle; basically we want to use our data to estimate model parameters, and the bigger the ratio of the size of the data set to the number of parameters we have to estimate, the more we can feel we have genuinely modelled something. When, as with some neural net enthusiasts, there are more parameters than data, we can only groan and shake our heads. It is a sign that someone's education has been sadly neglected. How much faith would you have in the neural net B of *Fig.1.7* being able to do a decent job of predicting new points, based as it is on only two data points? How would you expect it to perform as more points are obtained? How much faith would you have in the rightness of something like B if the dimension were 99 instead of 2, more faith or less?

In the first chapter I remarked that the image on my computer screen could be regarded as a point in a space of dimension nearly four million, and that I didn't assert that this was a good idea. Suppose you wanted to write a program which could distinguish automatically between television commercials and monster movies. Doing this by trying to classify ten second slices of images using a neural net is something which might be contemplated by a certain sort of depraved mind. It would have to be pretty depraved, though. I shall return to this issue later when discussing model complexity.

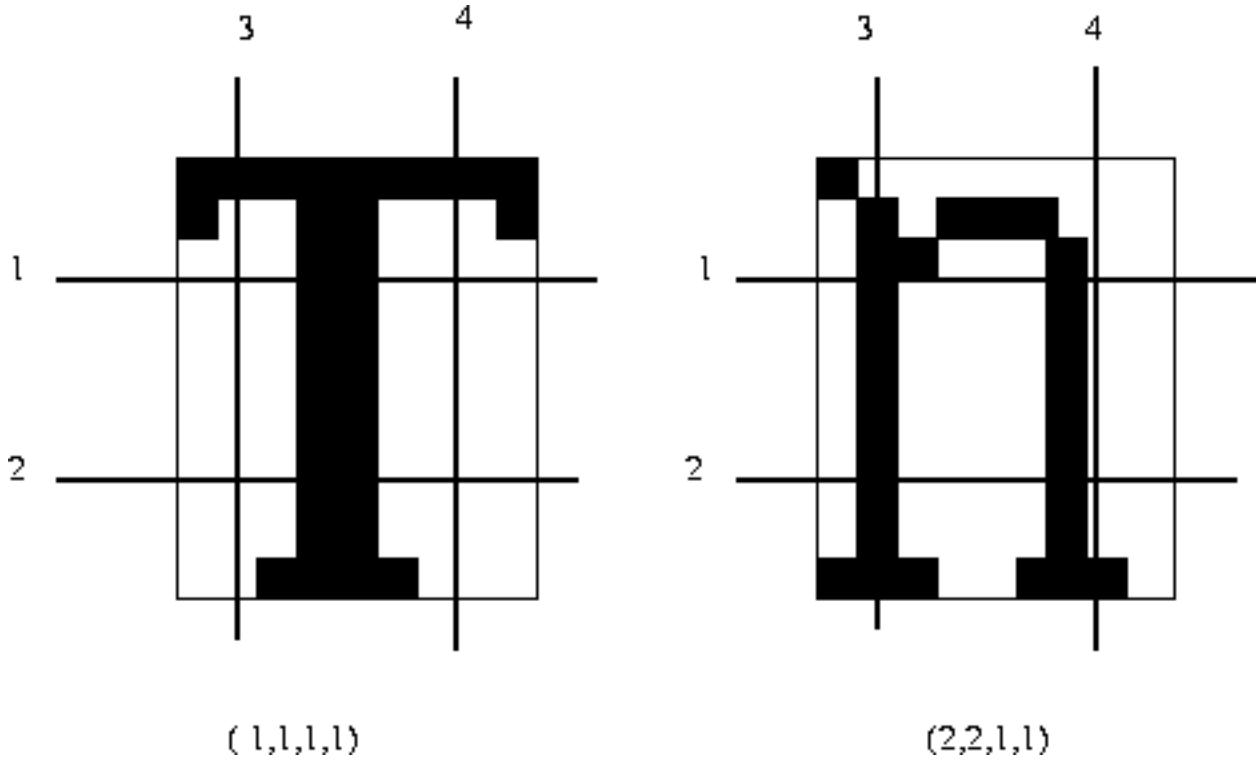
Next Up Previous Contents

Next: Scanline intersections and weights **Up:** Measurement practice **Previous:** Measurement practice

Mike Alder
9/19/1997

Scanline intersections and weights

Figure 2.7: A coding of two characters by measuring scan line intersections



Some thought on how to represent the characters in a way which will not be too badly affected by such things as translations, the deck transforms that are produced by italicisation and small angle rotations, may lead to taking horizontal and vertical scan lines across the rectangle, and measuring the number of distinct intersections. Or we can take the quantity of black pixels along a scan line, and list the resulting numbers in some fixed order.

For example, the characters in *Fig.2.7* have been coded by listing the number of intersections along two horizontal and two vertical scan lines, as indicated, making it easy to distinguish them as vectors in $\mathbb{R}^4$.

The method is not likely to be satisfactory with only two horizontal and two vertical scanlines, but increasing the number can give more information. A little ingenuity can readily suggest variants of the method which may even give some degree of font invariance.

Mike Alder
9/19/1997

Moments

Suppose we have isolated a character as a set of pixels. Let us not draw a box around it, let us first find its centre of gravity. If A is the set of pixels in the figure, with integer co-ordinates x and y for each pixel p in A, then let f_A be the characteristic function of A, i.e. $f_A\begin{pmatrix} x \\ y \end{pmatrix} = 1$ iff there is a pixel of A at location $\begin{pmatrix} x \\ y \end{pmatrix}$,and otherwise f_A takes the value 0. Then we can write the count of the number of pixels in A as

$$\sum f_A\begin{pmatrix} x \\ y \end{pmatrix}$$

where the summation is taken over the whole plane of integer value pixel locations. We can write the mean of the x values of all the pixels in A in the form:

$$\bar{x} = \frac{\sum x f_A\begin{pmatrix} x \\ y \end{pmatrix}}{\sum f_A\begin{pmatrix} x \\ y \end{pmatrix}}$$

and similarly

$$\bar{y} = \frac{\sum y f_A\begin{pmatrix} x \\ y \end{pmatrix}}{\sum f_A\begin{pmatrix} x \\ y \end{pmatrix}}$$

where sums are taken over the whole plane. (Since f_A is zero over all but a bounded region, we don't really have to do as much arithmetic as this suggests.)

These numbers give us information about the set A. We can rewrite things slightly by defining the $(p,q)^{th}$ *moment* of f_A (or of A) by

$$\mu_{[p,q]} = \sum x^p y^q f_A \begin{pmatrix} x \\ y \end{pmatrix}$$

for any natural numbers p,q,

and the *normalised* $(p,q)^{th}$ *moment* of f_A (or of A) by

$$\mu_{[p,q]} = \frac{\sum x^p y^q f_A \begin{pmatrix} x \\ y \end{pmatrix}}{\sum f_A \begin{pmatrix} x \\ y \end{pmatrix}}$$

for any natural numbers p,q,

Then the moment $\mu_{[0,0]}$ is the pixel count of A,the mean x-value of A is the moment $\mu_{[1,0]}$ divided by the pixel count,or alternatively the normalised (1,0) moment, and the mean of the y-values of the pixels of A is the moment $\mu_{[0,1]}$ divided by the pixel count.

It is easy also to compute higher order moments. These give extra information about the distribution of the pixels in A.

The *central moments* are computed in the same way, except that the origin is shifted to the centroid $\begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix}$

for all the pixels of A.

All the moments $\mu_{[p,q]}$ where p+q takes the value v, are called *moments of order v*. A little thought and recollection of statistics will no doubt remind you that the *second order central moments, moments of order 2,* i.e. the (2,0), (0,2) and (1,1) central moments, are the elements of the covariance matrix of the set of points, except for a scaling dependent on the number of points. The use of central moments is clearly a way of taking out any information about where the set A actually *is*. Since this is a good idea, we have a distinct preference for the central moments. The three central moments of order 2 would allow us to distinguish between a disk and a thin bar rather easily. In the exercises we ask you to compute the central moments for some simple shapes.

Example We give an example for the easy case of sets A and B defined by

$$A = \{\begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{Z}^2 : -5 \leq x \leq 5 \ \& \ -1 \leq y \leq 1\}$$

and

$$B = \{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{Z}^2 : -3 \leq x \leq 3 \ \& \ -2 \leq y \leq 2 \}$$

Then A has 33 pixels and B has 35; B is squarer than A. Both have the origin as the centroid so as to save us some arithmetic.

A has three rows, so to calculate $\mu_{[2,0]}(A)$ which is the sum of the squares of all the x-values of the pixels in A we get

$$3(-5)^2 + 3(-4)^2 + 3(-3)^2 + 3(-2)^2 + 3(-1)^2 + \cdots + 3(5)^2$$

i.e. $\mu_{[2,0]}(A) = 330$. Similarly, $\mu_{[0,2]}(A) = 26$, and

$$\mu_{[1,1]}(A) = (-5)(-1) + (-4)(-1) + \cdots + (4)(-1) + (5)(-1)$$
$$+(-5)(0) + \cdots + (5)(0) + (-5)(1) + \cdots + (5)(1) = 0$$

So the three second order moments, in the right order, give the vector $\begin{pmatrix} 330 \\ 0 \\ 26 \end{pmatrix}$, while the same calculation for B gives $\begin{pmatrix} 140 \\ 0 \\ 70 \end{pmatrix}$.

So if we were to represent these objects as points in $\mathbb{R}^3$, it would be easy to tell them apart. Unfortunately, it takes rather higher dimensions, i.e. higher order moments, to discriminate between characters, but it can be done by the same methods.

Next Up Previous Contents

Next: Zernike moments and the **Up:** Measurement practice **Previous:** Scanline intersections and weights *Mike Alder*
9/19/1997

Zernike moments and the FFT

The expression for the calculation of the moments in terms of a sum would be written by a mathematician in the more general form of an integral. This covers the case for a density function over the plane, or indeed over $\mathbb{R}^n$ for any positive integer n as well as the discrete case. The density function does not have to a characteristic function, we could allow grey levels, so f_P can be supposed to be an integrable function in $\mathbb{R}^n$, although we shall mainly be interested in the case $n = 2$. Integrating most functions over the whole plane does not give sensible answers, so we assume that the functions are defined only in the unit disk, $D^2 = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 \leq 1\}$. We can normalise the regions of interest to us by scaling down its size until it just fits into the unit disk.

If we write for Natural Numbers p,q

$$\mu_{[p,q]} = \int_{D^2} x^p y^q f_A \begin{pmatrix} x \\ y \end{pmatrix}$$

it is clear that the number $\mu_{[p,q]}$ is the result of taking the inner product of f_A with the monomial $x^p y^q$ in the space of real valued functions from D^2, otherwise $\mathcal{L}_2(D^2)$. (If you need to brush up your linear algebra, particularly in respect of function spaces, see Kreider, Kuller, Ostberg and Perkins, subsequently KKOP for short, in the bibliography at the end of this chapter. It is a *good* book, and there aren't very many such.) This is precisely the same idea as expressing a function by its Fourier coefficients, except that instead of using the sine and cosine functions, we have chosen to use monomials.

This leads to the obvious question as to whether there is a better basis of functions to use in order to express the characteristic function of a shape. In particular, why not take the Discrete Fourier Transform of the image? This may be done quickly and efficiently using the

FFT or Fast Fourier Transform, and is described in Image Processing courses. Or are there better bases of functions, more suited for the shapes we get in character recognition?

A small amount of reflection on what you are doing when you do a two dimensional Fourier expansion of the characteristic function of a character suggests that this is not a particularly good way to go. Imagine doing it in one dimension: you have a set which is a couple of intervals and you have the characteristic function for the set, which looks like a rectangle, 1 over the set, 0 elsewhere. You approximate this by adding up sine and cosine waves until you get a reasonable fit. Now go to two dimensions and think of a character, say the letter /E/. You have a function which is of height 1 over the

pixels in the character, and 0 outside the character. Now you want to build this function of two variables up out of sines and cosines in the x direction and in the y direction.

Closing your eyes and brooding about it quietly, leads you to the suspicion that it will be a messy business with rather a lot of terms before anything much like the /E/ emerges from the mess. Of course, the same considerations suggest using polynomials would be a mistake too, but at least it is easier to do the arithmetic for polynomials.

On the other hand, the trigonometric functions do have one advantage, they form an orthogonal set of functions, which makes the set of coefficients much more informative. So we might ask whether we can do anything to improve on the monomials.

In KKOP, page 277, the reader is shown how to use the *Gram-Schmidt Orthogonalisation Process* to make the sequence of basis functions

$$1, x, x^2, x^3, \cdots$$

orthogonal on the interval [-1,1], thus producing the *Legendre Polynomials*. It is a straightforward application of linear algebra ideas to spaces of functions, something all serious scientists and engineers should know. (If you were brought up to believe that engineers could memorise the recipes and not trouble to figure out why they worked, you should sue your educators for your fees. You wuz robbed. There are programs that can do what you were trained to do, and they probably do it better. So why should anyone hire you, when he can buy a program cheaper?)

If we normalise all our shapes so that they are scaled to lie in the unit disk in the plane, and have their centroid at the origin, and if we use polar co-ordinates, we can expand any shape in the plane into the set of basis functions, indexed by the natural numbers p,q:

$$V_{p,q} = r^p e^{jq\theta}$$

where *j* denotes the square root of -1, and (r, θ) are the polar co-ordinates of points in the unit disk,

$$D^2 = \{ \begin{pmatrix} r \\ \theta \end{pmatrix} : 0 \leq r \leq 1 \ \& \ 0 \leq \theta < 2\pi \}$$

The result of applying the Gram-Schmidt Orthogonalisation Process to the above basis of functions gives us the *Zernike basis*, $Z_{[p,q]}$.

Although it is fairly unpleasant to compute these by hand, some programming in a good symbolic package such as MAPLE or MATHEMATICA or MACSYMA helps. Alternatively one can get closed form expressions for the functions by reading the right papers. See Alireza Khotanzad and Jiin-Her Lu in the bibliography. The former give the formula:

$$Z_{[p,q]}(\rho,\theta) = R_{[p,q]}(\rho)exp(jq\theta)$$

where

p	is a positive integer or zero,
q	is a positive or negative integer, subject to the constraints (p-$\|q\|$) is even and $\|q\| \leq p$,
ρ	is the distance from the origin to the pixel,
θ	is the angle between the radius line to the pixel and the x-axis,
$R_{[p,q]}(\rho)$	is the radial polynomial defined by:

$$R_{[p,q]}(\rho) = \sum_{s=0}^{(p-|q|)/2} \frac{(-1)^s\,[(p-s)!]\rho^{p-2s}}{s!(\frac{p+|q|}{2}-s)!(\frac{p-|q|}{2}-s)!}$$

Note that $R_{[p,-q]}(\rho) = R_{[p,q]}(\rho)$.

Then the *Zernike moment of order p* with repetition *q* for a function *f* defined in the unit disk (possibly the characteristic function of a set) is

$$A_{[p,q]} = \frac{p+1}{\pi}\int_0^{2\pi}\int_0^1 f(\rho,\theta)Z^*_{[p,q]}(\rho,\theta)\rho d\rho d\theta.$$

The obvious changes are made to the digitised case, replacing the integrals by sums. * denotes the complex conjugate.

Projecting down on this basis in the space $\mathcal{L}_2(D^2)$, the set of square integrable functions defined on the interior and boundary of the unit disk in $\mathbb{R}^2$, or the discrete pixelised approximation to it, gives all the *Zernike moments*. They are complex numbers in general.

We have ensured that the objects we are going to expand in this way have a representation which is shift invariant by reducing it to the centroid, and scale invariant by normalising it to lie inside the unit disk. If

we observe that the expansion in terms of the basis functions expressed radially will have a complex number arising out of the angular component, then by simply taking the modulus of this number we ensure that the resulting coefficients give a representation of the shape which is rotation invariant as well. Naturally, this leads to confusing commas and sixes and nines and single quote marks. The measurement suite has rather more invariance than is good for it, but it is a simple matter to add in a few extra terms which tell you where the character is, how big it is, and which way is up.

Experimentally it is found that it is feasible to recover almost complete information about the shape of a point set in $\mathbb{R}^2$ using less than 20 terms in the infinite collection of moments, when the point set is something of complexity comparable with a printed character. The general use of moments is favoured because, happily, only a small number of moments often suffice to give adequate approximations.

- Historical Note

Next Up Previous Contents

Next: Historical Note **Up:** Measurement practice **Previous:** Moments *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Masks and templates **Up:** Zernike moments and the **Previous:** Zernike moments and the

Historical Note

I have looked up the 1934 paper in *Physica* by Zernike. In the same issue is a paper by Van der Pol, for those of you who remember his oscillator. I cannot find any reference in Zernike's paper to the moments that bear his name. Mind you, its in German, so I may be mistaken. There are Bessel functions and Gamma functions galore, but not the Zernike basis. Since you can expand in more or less any old functions, using the functions given above can't do any harm, but since I haven't verified orthogonality in D^2, I don't know that this is actually correct. You can do it for me, quite quickly.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Invariants **Up:** Measurement practice **Previous:** Historical Note

Masks and templates

The earliest way of reading characters automatically was to look at each character through a family of masks, basically holes in a piece of metal, and see how much was visible in each case.

This is not essentially different from measuring intersections with scan lines, except that the mask holes don't have to be lines, they can be any shape. Nor is it very different in principle from *Exercise 1.6.3*, where we have weights of 1 and 0 on the arcs joining input cells to the neural unit. The older terminology also leads to terms like *template matching*, where you have a mask exactly the shape of the character you are trying to detect, and you measure, in effect the distance from the template mask. To give an example, suppose you wanted to detect a vertical bar in the middle of a three by three grid of cells, say a white bar on a black background. Then you could arrange the weights so as to give a zero to every cell of a three by three array of cells where the actual cell value was the same as the desired pattern, and a plus one to every cell where there is a difference. Then this simply gives as sum the Hamming distance between the two images. If zero, they are identical, if 9 they are negatives of each other. Or you can change things round so that the score is plus 1 when they are the same and zero when they differ, which means you go for the high score instead of the lowest. The difference between the two systems is rather trivial. So masks and templates are just old fashioned language for measuring distances between points in $\mathbb{R}^n$ where n is the number of pixels in the image and the entries in the vectors are usually just 0 or 1.

It is worth being clear about this. You measure similarity between image and template, you measure a distance in some representation space. If you do this for several templates, you get a vector of distances. That gives a new representation in which your choice of the `right' interpretation is just the vector component which is smallest. This is the metric method discussed in chapter one.

Next Up Previous Contents

Next: Invariants **Up:** Measurement practice **Previous:** Historical Note *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Chaincoding **Up:** Measurement practice **Previous:** Masks and templates

Invariants

It is clearly desirable to select between different methods carefully on the basis of the ease of final classification. We can say something about this at an early stage: if there is a method of getting from a pixel array to a vector of numbers which is invariant with respect to the transformations which occur in images of characters, for example, shifting, scaling, the `deck transform' that goes from normal font to italic, then it is probably more likely to give a satisfactory result than one which isn't. So it is worth asking, for each measurement scheme, what are its invariants, and are the transformations which get you from one character to a different version of `the same' character going to preserve them? It is intuitively obvious that if I could give a `template' character for a /5/ say, and I could list all the ways of transforming this particular point set into every other point set representing a /5/ and to no other point set, then I would have a way of deciding if something is a /5/ or not. If I have some coding into vectors (or anything else) which is *invariant* under these transformations and only these, then I have a good way of telling a /5/ from anything else. This way of looking at things may seem bizarre at first encounter but is rather powerful. It has proved its worth in Physics.

This has led some workers to look for topological invariants, presumably in a spirit of pure enquiry. The topological invariants are very basic indeed, since in topology we allow some very considerable stretchings and deformations of an object: it has been said that a topologist is the sort of man who can't tell a coffee cup from a doughnut. The reason is that you can transform one continuously into the other, if you don't mind looking rather silly when the handle falls off your doughnut while you are using it to hold coffee.

The only topological invariants for characters of the English language are the number of holes in them, and the number of components in a character (two for /;/, for instance). So hole counting allows one to distinguish /1/ from /0/ from /8/, but not /1/ from /5/ or /4/ from /9/. This is (a) fairly useless and (b) can go badly wrong with quite small amounts of noise, which can turn an /S/ into an /8/ or /9/ with no great difficulty. The use of topological invariants therefore stems from a misunderstanding of why invariants are important. The most important transformation between versions of the `same' character is frequently that of adding noise, which is not a topological transformation at all. So any measuring process, any procedure for getting from pixel sets to vectors, which is corrupted by adding noise is not an attractive bet.

Next Up Previous Contents

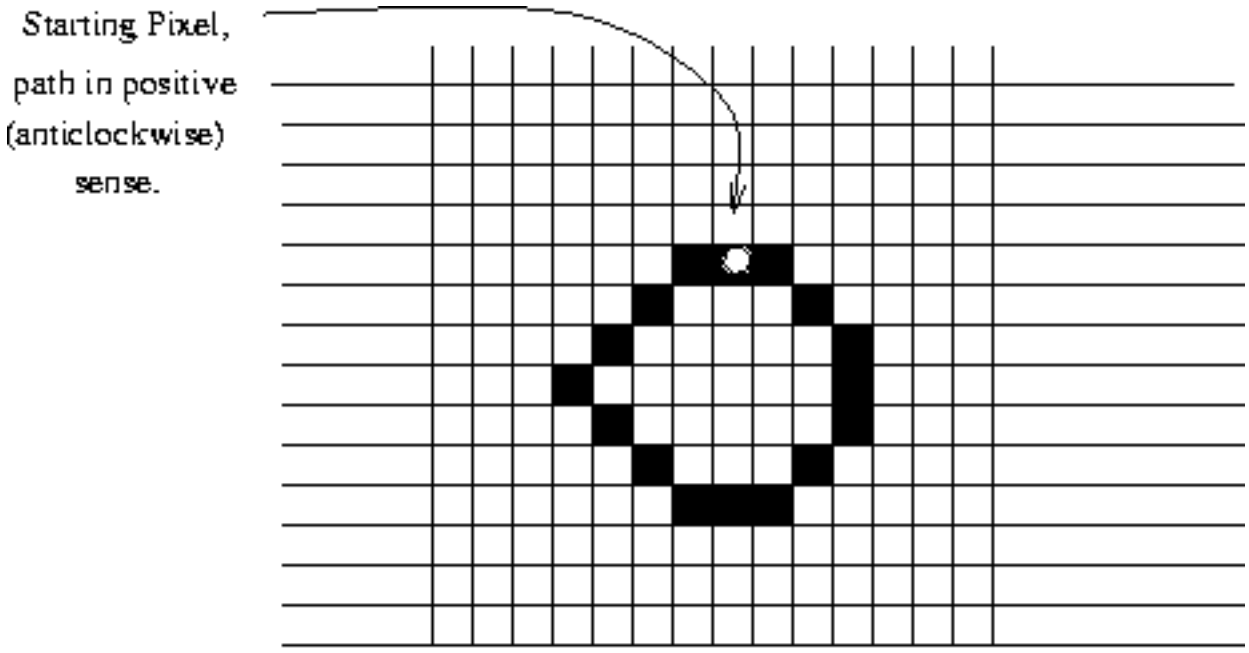
Next: Chaincoding **Up:** Measurement practice **Previous:** Masks and templates *Mike Alder*
9/19/1997

Chaincoding

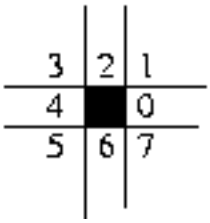
There are some mildly intelligent things one can do to reduce the computational load in getting from an image to a vector of numbers which describes the image. One is to reduce the number of pixels we have to put into computations by ignoring the interior and simply looking at the boundary. This will only work if the noise levels are low, since reducing the number of pixels will probably put up the Signal to Noise ratio. If the boundary is corrupted, it won't matter so much if we have a lot of other pixels used in estimating the shape. On the other hand, there are ways of tracing the boundary and then smoothing it in case the odd pixel has been tacked on or gouged out, and since we can tell each character by looking at its boundary and boundaries have fewer pixels, some amount of boundary smoothing followed by an analysis of the boundary shape may be an attractive option in some cases.

A boundary is a relatively simple set and can be specified by *chain code*, where we pick a starting pixel and then specify the direction of the next by labelling the adjacent pixels, as in *Fig. 2.8*.

Figure 2.8: Chain Coding of a boundary.



Chain Code is: (4,5,5,5,7,7,7,0,0,1,1,2,2,3,3,4)



Either by such a coding or by linescan methods, it is possible to do such things as count convexities. If circumnavigating a letter /E/ for example, the changes of direction can be determined, and much useable information can be extracted. Many variants of this will occur to the ingenious student. Smoothing boundaries is properly part of an image processing course and there are many ways of doing this; an erosion followed by a dilation can accomplish something of this.

Again, much depends on the kind of data with which one is confronted, but hand written characters are harder to process in this way than printed characters; there is less consistency in general. For printed characters, it is possible to segment the boundary, smooth it, identify strokes and generally extract complex relationships. These methods are not generally robust under change of font, still less do they work for handwritten characters.

Next Up Previous Contents

Next: Syntactic Methods **Up:** Measurement practice **Previous:** Invariants *Mike Alder*
9/19/1997

Syntactic Methods

The line of thought which starts off with segmentation of a page of text into lines, then into characters by checking that the set of points is separated by white pixels from other sets, and then tries to recognise the characters by looking at their shape, is liable to several sorts of criticism. Contemplation of *Fig.2.3*, the postcode digits, suggests that the shape based methods might work well with printed characters where there is a high degree of consistency and where, except when noise levels get very high, characters are isolatable by finding boundaries. Up to a point this is correct, as some experimenting with the exercises will show you. On the other hand, for hand-written characters, there is something unsatisfactory about this whole approach. As a person who writes with a pen, I am conscious that I make strokes with more or less care and attention; that the changes of direction are pretty important, but that once I have embarked upon generating a stroke, the thickness is of minor significance. I can easily tell by looking at *Fig.2.3* that the horizontal stroke belongs with the second character, i.e. is part of a /5/ and not of a /7/. I have no trouble about the lower end of the /5/ intruding into the /6/, nor the /2/ intersecting the horizontal stroke of the /5/.

Some sort of syntactic processing is going on here, whereby the image is decomposed by the eye not into pixels and then into characters in one step; there is an intermediate entity, the stroke, made up out of pixels, and itself a component part of a character. The eye has no difficulty joining the two components of the /5/; it expects a horizontal stroke because of the other two strokes making up the /5/. If the horizontal stroke were moved two centimetres to the left, it would have been a /3/, but it has to be one or the other. And so the arrangement of some of the strokes leads to hypotheses about the remaining strokes, hypotheses which are confirmed and lead to identifications of other elements.

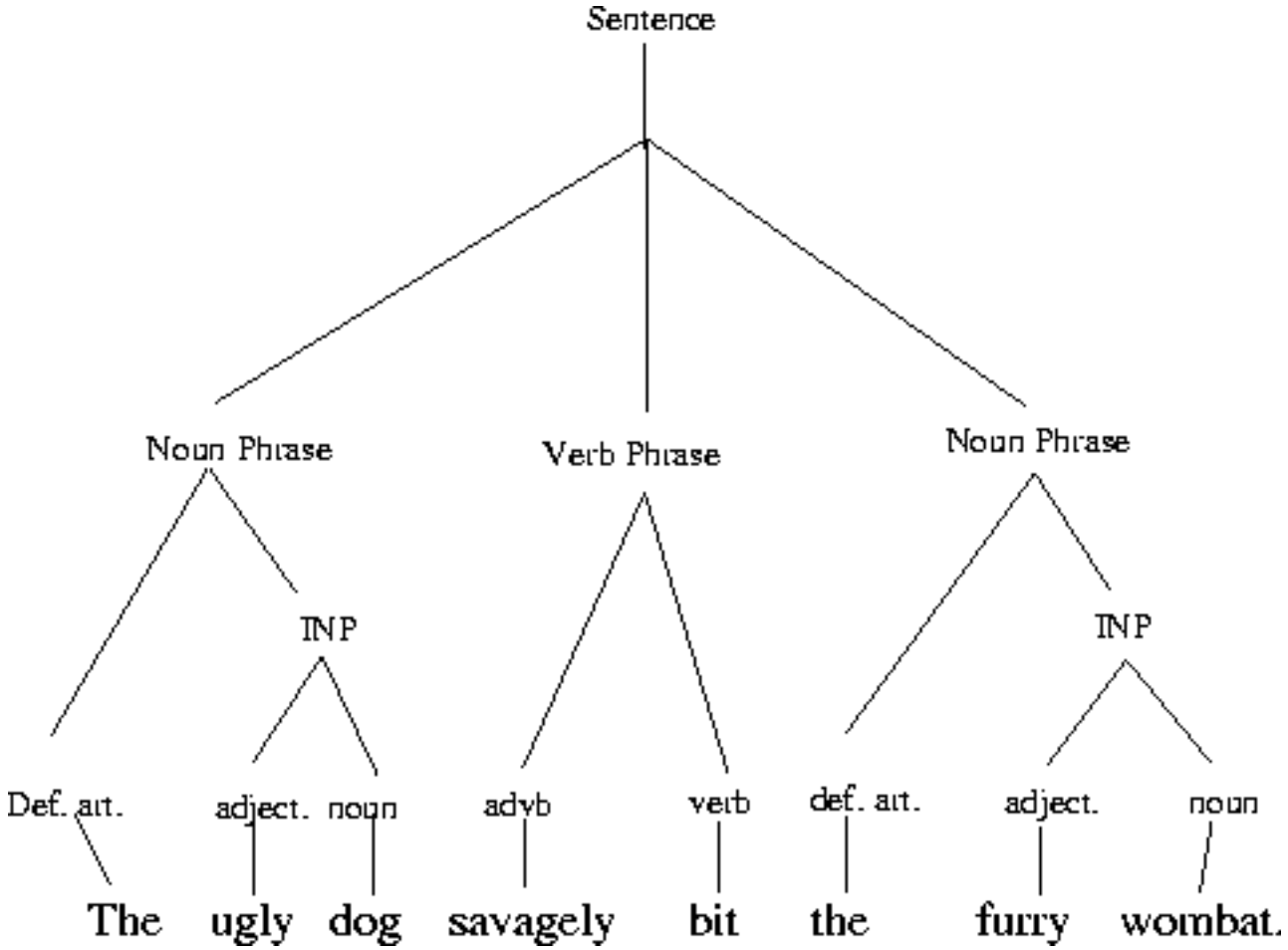
Attempts to articulate this view of optical pattern recognition have tended into a quagmire. The Artificial Intelligentsia have tried (in such programs as HEARSAY, from Carnegie Mellon University) to articulate the notion of competing hypotheses resolved by some higher level adjudicator, but their methods of representing hypotheses have been mathematically naive and have had only limited success. To be fair, the problem is far from trivial.

What is known of the physiology of the the lower level processing in the mammalian eye leads us to the notion of a hierarchy of levels, proceeding from essentially pixels, retinal cells triggered by rhodopsin release, to edges or blobs or motion. It is known that a layer away from the retinal cells, there are cells which respond to edges in specific orientations. It is not too hard to believe that a little further back again there are cells which respond to strokes, sequences of edges. And so on, until, conceivably, there are cells responding to digits.

Whether this continues to the level of the mythical `grandmother neuron' which fires when and only when you see your grandmother, is a matter into which we shall not enquire. What is plausible from neurophysiology is that some aggregation process builds `molecules' out of `atoms', and then supermolecules out of molecules. In the same way, one might say, as letters aggregate to words, and so

on. By analogy with the structure of language, whereby a sentence is considered to be derived from more primitive entities by means of rewrite rules, we talk of a *grammar* and of *syntax* of the components.

Figure 2.9: The generation of a sentence of words by applications of rewrite rules



The diagram in *Fig.2.9* is, more or less, a *parse* of a sentence in terms of a phrase structure grammar, showing how each symbol (like `sentence', Noun phrase') at any level from the top down, is rewritten to a string of symbols at the level below it, terminating in symbols which are words of the English language. Actually, we could go one level further down into a string of letters; we don't do this because meddling grammarians a few centuries ago regularised the spelling of words, so there is only one way to spell a word, which makes the rewrite rather uninteresting. Around Shakespeare's day, when an English Gentleman spelled the way he fancied and took no nonsense from grammarians, there was a choice. Winston Churchill was notorious for being atavistic in this and other respects, and if your spelling is not all it should be, console yourself with the thought that you have an eminent rôle model.

The question `how can one decompose objects like *Fig.2.3* into strokes and then characters?' leads us into interesting places which will be explored more thoroughly when we get to Syntactic Pattern Recognition. The motivation for doing so is worth reflecting on; strokes are produced by people and recognised by people as comprising characters. This is true not just in European scripts of course; Chinese characters and Japanese characters as well as Arabic and the many other scripts are usually decomposed in this way.

Cuneiform is not, but you don't see a lot of *that* about these days. The idea that it might be good policy to unravel the problem of reading characters by studying the processes at work in human recognition methods produces interesting results upon which I shall expand later.

Next Up Previous Contents

Next: Summary of OCR Measurement **Up:** Image Measurements **Previous:** Chaincoding *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Other Kinds of Binary **Up:** Image Measurements **Previous:** Syntactic Methods

Summary of OCR Measurement Methods

So far in this chapter we have focussed on the recognition of characters, principally on printed characters, and we have been exclusively concerned with getting from a point set in the plane to a vector of real numbers which describes that set. The first difficulty is the problem of segmenting an image of text into separate characters, and I remarked that this approach, in which the character is regarded as the pixel set of interest, cannot be expected to work well for cursive hand-written text, suggesting that the methodology needs attention. We then examined three methods of turning the point set into a real vector. The first was to put the character into a box with a fixed number of pixels in a rectangular array (after some transformation), and then raster scan the array. This method was not recommended. The second was to apply a family of masks to the set and extract information about intersections between the mask and the pixel set. And the third was a family of methods which included taking a Fourier series expansion as a special case, but is more commonly described as the use of moments. This entails computing inner products in the function space $\mathcal{L}_2(D^2)$, and hence projecting down onto a basis for the set of square integrable functions defined on the unit disk. Choosing polynomials in the radius and trigonometric functions in the angle and orthogonalising gives us the *Zernike moments.*

In deciding between different methods, we were much concerned with the problems of having the representation *invariant* with respect to the kinds of transformations which occur in practice with characters in text- shifting, scaling and the `deck transformation' were mentioned, as was rotational transforms. Making the system robust, or invariant under low levels of noise, can also be thought of as part of the same framework of looking at the problem.

It was observed that these methods could be applied to just the boundary set with some possible saving in computation but at the risk of being more vulnerable to noise.

Specialised methods arising from boundary tracing exist, such as fitting functions to the border of a character and counting convexities, curvatures, or other quantities.

We concluded by observing that the possibility of going through intervening steps, so that a character should more usefully be regarded as being built up out of *strokes*, and strokes being built up possibly out of something else, giving a hierarchy of structures and substructures, was attractive as a generic model for human information processing. Again, there were promises to keep.

Next Up Previous Contents

Next: Other Kinds of Binary **Up:** Image Measurements **Previous:** Syntactic Methods *Mike Alder 9/19/1997*

Other Kinds of Binary Image

I have gone to some trouble to discuss binary images of characters because it is now useful to ask oneself, which of the methods discussed can be expected to generalise to other kinds of image? For binary images, the answer is most of the techniques mentioned are still useful.

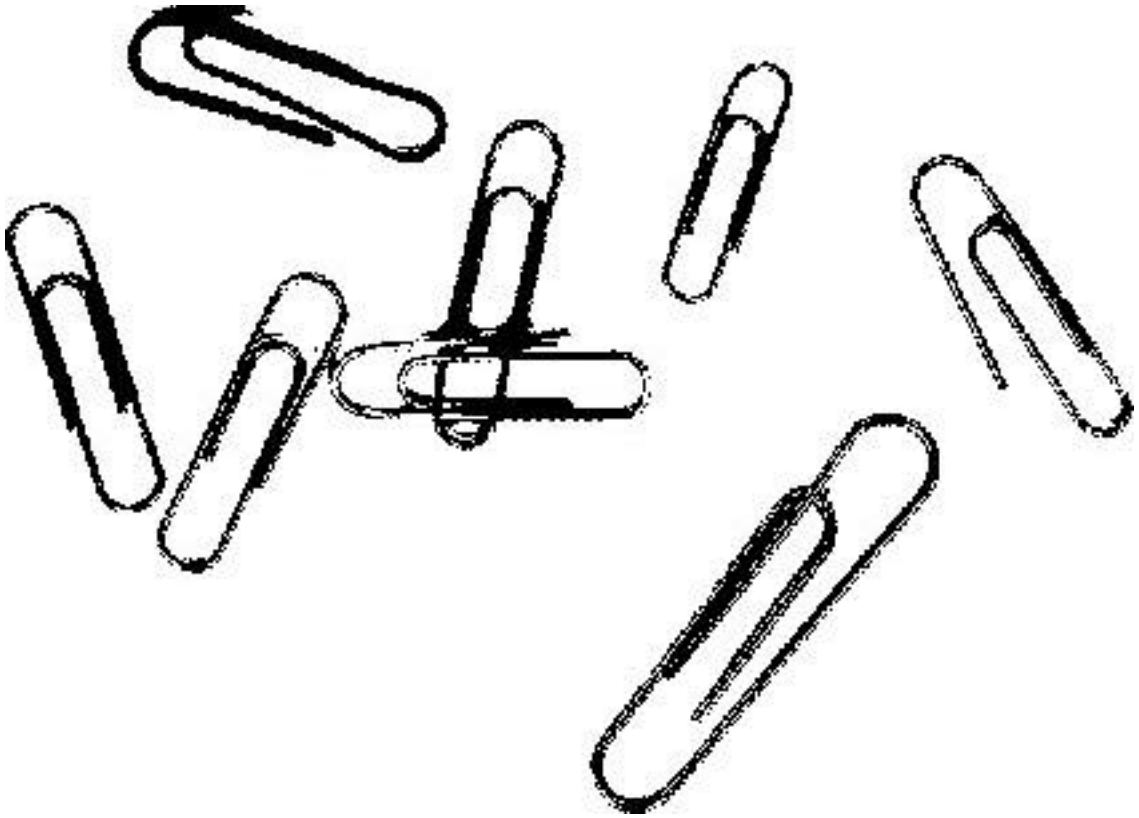
Binary images, or sufficiently good approximations thereto to allow thresholding to binary to give reliable results, arise in the world naturally, and range from bar codes which are designed to be machine readable and give problems to people, to cartoon sketches of politicians the recognition of which by human beings is just this side of miraculous and by machine currently out of the question.

Handprinted characters, signatures, writing on cheques and the results of filling in government forms, all can be captured as, or thresholded to, binary images.

The problem of segmentation arises whenever it makes sense to say there are several different objects in an image, and they are disjoint. Of course, this need not happen at all. Images obtained by pointing a camera at machine parts or other physical objects for the purposes of sorting or counting them can easily look like *Fig.2.10.*

Figure 2.10: Binary image of real objects (see over)

Here, the rotational invariance of measurements is of much greater practical importance than with images of characters which tend to come reasonably consistently aligned. The spacings are not as consistent as with printed characters, and may not exist at all. (Some OCR enthusiasts have been known, in their cups, to bemoan the passing of the typewriter, which produced output which is much easier to read automatically than properly printed material. This is in conformity with Alder's Law of artificiality, which states that if it is quick and easy for a machine to do something, then whatever it is will be pretty damned ugly.)



Preprocessing by morphology techniques is commonly used; we erode the objects until they are separated, and this allows us to count them and isolate them. This can be something of a failure, as with biological images of, say chromosomes, included as a .tif file on disk, or the paper clips in *Fig.2.10*. Erosion can split objects into two parts, particularly objects which have narrow waists, and naturally fails with high degrees of overlap as in the case of the paper clips. I shall have more to say about the segmentation problem in the chapter on Syntactic Pattern Recognition, where it plainly belongs. It is clear to the reflective mind that we do not in fact recognise handwritten words by first identifying the letters and then recognising the word. We do not read that way. We recognise the words before we have recognised all the letters, and then use our knowledge of the word to segment into letters. It ought to be possible to do the same thing with images of objects having regular structure, or objects which have a multiplicity of occurrences with more or less the same form. In particular, the human eye has no problem with counting the number of paper clips in the image of *Fig.2.10*, but a program has to be written which contains information about the shape of a paper clip. Even a Trobriand Islander who has never seen a paper clip in his life has no great difficulty counting to eight objects instead of six or seven. It is plain that information is extracted from part of the image in order to decide what constitutes an object, and this information applied elsewhere. This is a clever trick which involves learning; how it may be implemented in a program will be discussed later.

For monochrome images with many grey levels, some techniques still survive as useful in specific classes of image. For full colour images, life is harder again, and except for rather simple images, or images which can be converted into grey scale images with only a few grey levels, we have to consider other methods.

I shall discuss particular images and their provenance and outline the kind of modifications which may still give us a vector of real numbers.

Next Up Previous Contents

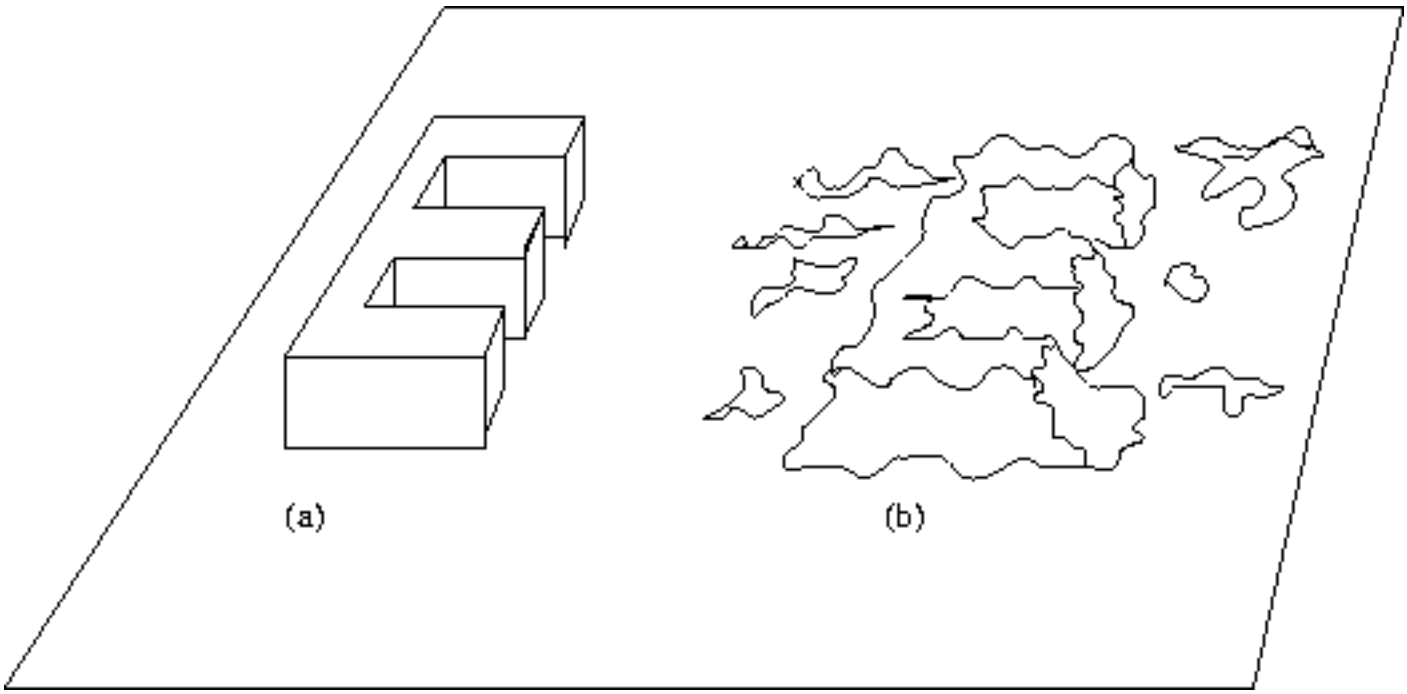
Next: Greyscale images of characters **Up:** Image Measurements **Previous:** Summary of OCR Measurement

Mike Alder
9/19/1997

Greyscale images of characters

It is common practice in manufacturing steel to stamp an identification code on each bar of the stuff. This is done with a metal stamp and a hammer or equivalent. The result may be several digits punched into one end of a bar of steel, and it would be nice in some applications to be able to read these digits automatically by pointing a camera at the bars as they roll by. Variants of this problem occur throughout industry; sometimes the only action taken is to make stock entries, sometimes other automation responses are appropriate. Sometimes, the stamping process is itself automated and needs to be verified.

Figure 2.11: A letter `E' regarded as a function; (a) binary, (b) grey scale



The image now is a grey scale one.
I am grateful to Chris de Silva and Gek Lim of CIIPS, The University of Western Australia, for image files and also many programs used in image processing.

The first thing to do with a greyscale image of a character, if at all possible, is to threshold it back to a binary image. Inspection of an image will, sometimes, allow you to decide if this

is feasible. In the case of the stamped characters, thresholding to a binary image does not work well here, since

information relevant to the recognition is lost. Ideally one would have a thresholding process which adapted the threshold to the local environment rather than one which merely set a value and stuck with it. This sort of consideration leads us to the theory of *filtering*, which is properly part of an image processing course rather than a pattern recognition book, and some pointers to the literature are provided in the bibliography at chapter's end. Removing `textural noise' can be done to some extent, but it still leaves us normally with a grey scale image.

In *Fig.2.11* we indicate two images, of a letter E in terms of the function from $\mathbb{R}^2$ to the brightness value in $\mathbb{R}$, one is a binary image and the other a greyscale image. It is useful to look at images as functions defined in some region of the plane in much of what follows.

- Segmentation: Edge Detection

Next Up Previous Contents

Next: Segmentation: Edge Detection **Up:** Image Measurements **Previous:** Other Kinds of Binary

Mike Alder
9/19/1997

Segmentation: Edge Detection

The segmentation of greyscale characters is rather more complex because we cannot just trace the white spaces between objects; the separating background will be textured. It makes sense to try to get a characterisation of the background texture and identify that. How this may be done will be outlined below.

Additional information may be available which can assist in segmentation, such as the number of characters and the alphabet. Life is simpler if we are looking for digits than for digits together with alphabetics, and if we know there are precisely four of them and how big they are, and possibly how separated they are, the problem of segmentation becomes easier. If I know that there is going to be a set of parallel texture bands of known width, orientation and number, distinguishable from the digits themselves at the texture level, then finding such separating bands may not be too hard a problem.

In general, the finding of edges in grey scale or colour images is a relatively complicated business; again this is properly part of an image processing course rather than a pattern recognition course, so I shall merely sketch the elements of the methods available.

Regarding the function which picks out a letter /E/ in *Fig.2.11*, we see that the bright regions (for I assume that we have here a bright letter against a black background) are the tops of mountains, and the dark regions are the background. For a black image on a white ground, the signs are reversed but no essential difference occurs. Now if an edge can be said to exist, it is a region where the gradient of the two dimensional function is steep. So we need to go around looking for the places where the gradient is high. This may be done by looking along a horizontal line and measuring the difference in pixel values for consecutive pixels. If this exceeds some threshold, you have a (vertical) edge at the pixel in the centre of the rapid change. Similarly one may look along vertical or diagonal lines. If you have found a possible edge by scanning left to right, some time may be saved by looking at adjacent pixels in the row underneath. Edges that have no extension in a direction transverse to the scan line are probably just noise. This method can be quite fast, and can be altered by making the threshold vary according to whether or not adjacent pixels in a preceding row or column have produced `micro' edges. This is a direct and obvious method which can work poorly on some images- but then, for each method there is an image it will perform badly on.

We can actually differentiate an image with respect to a direction, horizontal vertical or either of the diagonals, by simply rewriting each pixel to the difference between consecutive values: if in a scanline we have pixels x_n followed by pixel x_{n+1}, then in the differentiated image, we put $y_n = x_{n+1} - x_n$ and the result is, approximately, the partial derivative of the function in the horizontal direction if x_n is to the left of x_{n+1}. Similarly we could partially differentiate in the vertical or diagonal directions with the obvious modifications. When one of the partial derivative values exceed some threshold, we have a putative edge. Real edges are putative edges that have some other putative edges adjacent to them, and the longer the

chain of putative edges, the realer the edge becomes. This intuitively appealing idea can be formalised and made respectable.

Making the above system slightly more complicated, we can express it in the following terms: run a step function $g(n)$ along each row of the image, x_n which for consistency ought to be written as $x(n)$. Let g be defined by $g(0) = -1$, $g(-1) = 1$, $g(n) = 0$ for $n \neq 0, -1$. Now define

$y(n) = \sum_{t=-\infty}^{\infty} x(n-t)g(t)$. Now those depressing infinities vanish since g is zero except at 0 and -1, so $\forall n, y(n) = x(n)g(0) + x(n+1)g(-1)$ which gives us the partial derivative in whichever direction n is increasing. Or at least, a discrete approximation to it.

The expression $\sum_{t=-\infty}^{\infty} x(n-t)g(t)$ is said to be the *convolution* of x with g, and so we have treated a one dimensional convolution which outputs a directional derivative. It is convenient to think of this as taking a sort of generalised `window' over the function x, and applying a kind of *weighted average* operation to x, weighting by g, in order to get y. If, for example we had $g(0) = 0.5$, $g(1) = 0.25 = g(-1)$, then we would be doing a smoothing operation to x. The essential idea is simple: we input a function x, and replace $x(t)$ by some linear combination of the values of x in a neighbourhood of t. The set of coefficients in the linear combinations is specified by a function g. It is distinctly reminiscent of Morphology operations, but linear.

In two dimensions, it is common to impose a three by three array of pixels on which g is defined; everywhere else it is supposed to take the value zero. In the nine squares, the window, g, can have any set of values whatever, although we often normalise to ensure that they add up to 1. Then we generate a new image by plonking the window down on the (p,q) pixel of the old image, summing over the neighbourhood of this pixel with the weights of g applied, and then taking the resulting value and assigning it to the (p,q) pixel of the *filtered* output image. Suitable choices of g can find edges in different directions, in essentially the same way as the example of partial differentiation. Three by three windows are on the small side, and larger windows are more common.

Working through a few cases by hand beats the idea into ones skull very rapidly, and you should explore the exercises at the end of the chapter.

Using bigger windows or to put it in another way, taking a convolution with a function having a bigger support, can give improved results at slightly higher cost in time. Image convolutions may be done in hardware on some image processing boards.

And *that* is your very brief introduction to convolution filters. For more details, see any good book on image processing, several are cited in the bibliography.

Next Up Previous Contents

Next: Greyscale Images in general **Up:** Greyscale images of characters **Previous:** Greyscale images of characters *Mike Alder*

9/19/1997

Next Up Previous Contents

Next: Segmentation **Up:** Image Measurements **Previous:** Segmentation: Edge Detection

Greyscale Images in general

- Segmentation
- Measuring Greyscale Images
- Quantisation
- Textures

Mike Alder
9/19/1997

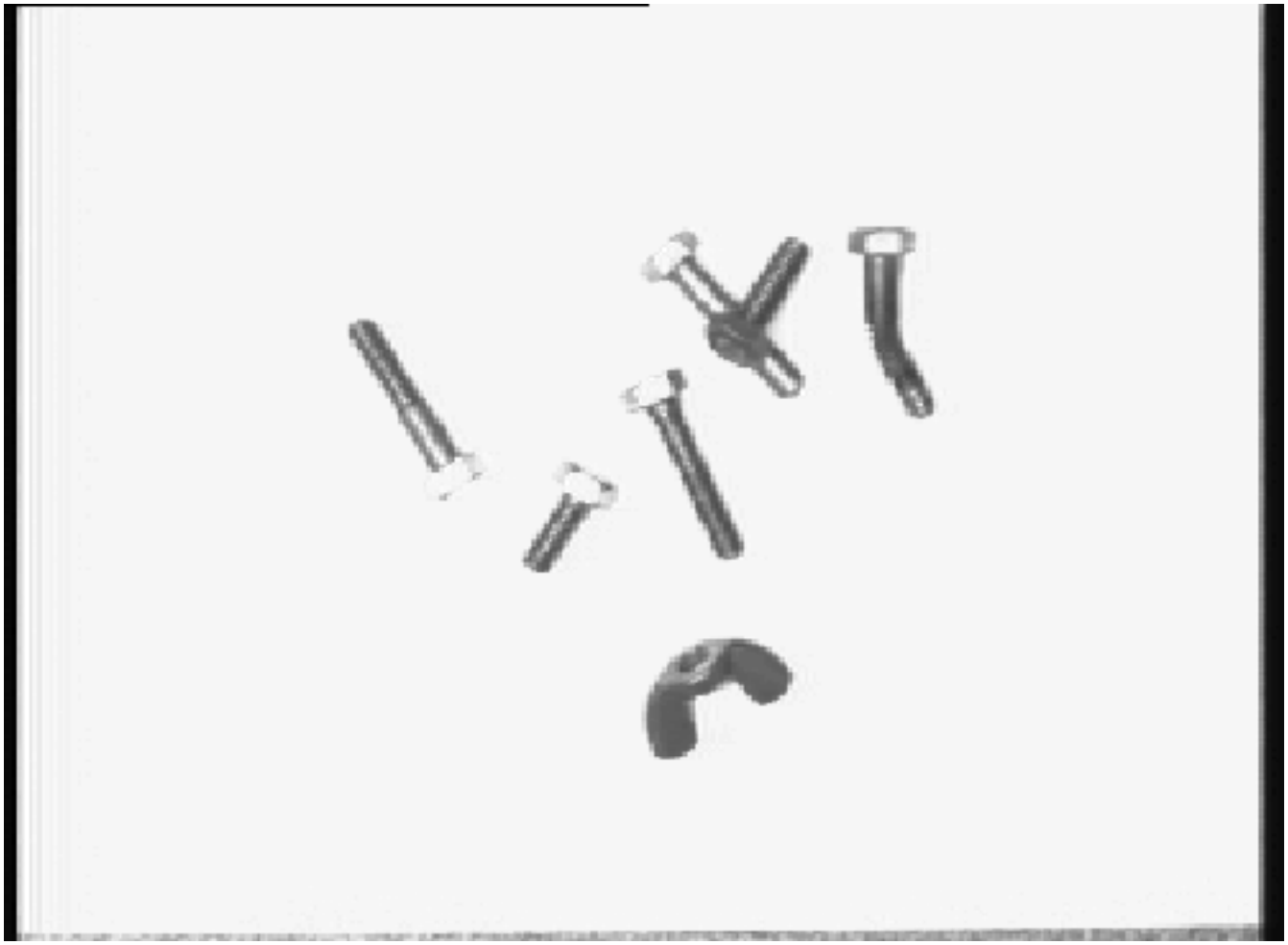
Segmentation

Everything said about greyscale images of characters holds for greyscale images in general, the extra degree of awfulness in segmentation is just as for binary images: there is less information about the extent to which things are separated and separation is not uniform in most applications. The very best thing to do with a greyscale image is to threshold it back to being a binary image if you possibly can. Unfortunately, this often gives poor quality results and more devious methods have to be used.

In *Fig.2.12* we have some grey scale objects, reproduced rather unconvincingly on a laser printer. It will give you some idea of what to expect on the *.tif* files supplied on disk. My thanks to Miss Gek Lim for producing *Fig.2.12* at very short notice.

Note that in this case we may not have a knowledge of what classes of objects are likely to turn up in the image, and that even the Trobriand Islander will have no difficulty counting seven objects, one of which is very different from the others, and two of which are overlapping. One of the six similar objects is bent, and one is short. One has the thread missing, but this is hard to see in the image on paper. This image was obtained by pointing a standard camera at the parts thrown down on a sheet of paper; although it has suffered in its many transmogrifications between camera and page, it is a higher quality image than others used in inspection in industry.

Figure 2.12: Nuts and bolts.



Mathematical morphology techniques can be extended to a limited extent to the greyscale case, see any of the books in the bibliography on the subject. Finding boundaries of objects by differentiating images sometimes works or can be made to work. There is no substitute for experience here, and if you do the exercises you will get plenty.

Next Up Previous Contents

Next: Measuring Greyscale Images **Up:** Greyscale Images in general **Previous:** Greyscale Images in general *Mike Alder*
9/19/1997

Measuring Greyscale Images

Let us suppose that, by finding edges instead of borders or by locating well defined spaces between the objects, we have succeeded in putting a box, perhaps an irregular quasi-border, around the object. The assumptions that went into measuring the `contents of a box' for binary images have to be examined anew for grey-scale images.

Transforms to normalise and register the image in some standard location proceed without essential

alteration. In the case where we are lucky enough to get regular boxes we may enclose our objects in standard rectangles if this looks to be plausible; much depends on what can be said *a priori* about the objects which may be found. If any edge boundary can be found, then at least the pixel array can be reduced to a centroid and brought inside the unit disc in $\mathbb{R}^2$. The computation of the centroid now has to be taken over real rather than integer values, but that is straightforward enough. Everything we learnt about moments is still applicable, except that f_A is no longer a characteristic function of the set, it is the function giving the pixel values. We may conveniently normalise these to lie between (black) and 1 (white). Moment methods are therefore popular and moderately robust in general.

It is useful to think of mask methods for binary images as an attempt to look at different regions of the image simultaneously, so as to do some parallel (or more accurately concurrent) processing. If

we take some shaped window and look at the image through it, we have the basic idea of mask processing. As a first development of that idea, we can measure what we see in a number of ways: one would be to collect a few central moments. Another would be to count regions, or pixels. Mask methods which measure the intersections with scan lines, or with some other kind of window on the image, can be just as effective with grey scale images as with binary images when they are estimating the total intensity in the field of the mask, but may fail due to incorrect thresholding when used to decide if something occurs or not. So the system of *Fig. 2.7* which simply counts intersections is not robust. What can be done is to extend the mask into higher dimensions: instead of regarding a mask as a sort of hole through which one looks at an image, a hole with its own characteristic function, one regards it as a continuous

function defined over a region, and then makes a comparison between the actual function which is the image, and the mask function. There are several different sorts of comparison which can be made. I leave it to the ingenuity of the reader to devise some: I shall return to the issue later in the book.

Mask methods tend to become, for computational convenience, convolutions at selected locations, and tend to be specific to the classes of objects to be classified. Fortunate is the man who can say in advance what kind of sub-images he is going to get after he has done some edge tracing. A basic problem with mask based methods is that it may be difficult to tell when you have something odd in the image which is different from anything your program has ever seen before.

The best way to find out about mask methods is to invent a few and see what they do. The exercises will

give you an opportunity to do this. Differentiating images is easy to do and perfectly intelligible after a little thought, and the results on greyscale images can be quite satisfying. See the disk files for some examples.

Next Up Previous Contents

Next: Quantisation **Up:** Greyscale Images in general **Previous:** Segmentation *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Textures **Up:** Greyscale Images in general **Previous:** Measuring Greyscale Images

Quantisation

Thresholding a greyscale image to a binary image is a particular case of reducing the range of levels. There are many applications where a reduction in the range has advantages. Quantisation over the greyscale is particularly important in image processing associated with compressing the image into fewer bits. A common technique of image compression is to first do a Discrete Cosine Transform of the image, which is just the even part of a Fourier Transform, and then take the resulting new image and quantise it judiciously. Then this quantised DCT image is stored. When the inverse DCT is performed, the original image is restored to quite high approximations, since the eye is relatively insensitive to very high and very low spatial frequencies. The same technique can, of course, be regarded as a measurement method.

Since one wants to make regions of the image which are close both in space and in grey level more likely to be assigned the same quantised value than regions of the image which are separate in either space or in grey level, it is convenient to work in the space of the graph of the function, as with *Fig.2.11*.

Again, this is more likely to be treated in a good book on image processing than one on Pattern Recognition, but the issue must be mentioned.

Next Up Previous Contents

Next: Textures **Up:** Greyscale Images in general **Previous:** Measuring Greyscale Images *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Colour Images **Up:** Greyscale Images in general **Previous:** Quantisation

Textures

It is worth while taking some colour pens and making strawberries, by the simple means of first colouring in a heart shaped region in red, and then going over it drawing lots of little green

`V' shapes. The resulting object, seen from a suitable distance, has a rather curious speckled look to it. The artist Cannaletto made a whole heap of money out of his discovery that you could make green paint look a bit like water if you painted lots of white ripples on top of it. Similar results can be obtained by playing with the background stipple patterns on the control panel of a Macintosh computer. Such a result is called a *texture* and may be obtained, as in the case of the Mac, with binary images.

If we take a rectangular grid and move it around the image, we find that the pattern seen at one location strikingly resembles the pattern seen when it is shifted by some more or less fixed number of pixels in any fixed direction. In general, this pattern is statistical rather than deterministic, but we can extract some statistics for the values in windows in this case too. For example, on a piece of reflective steel, upon which some characters have been incised, there is no regularity apart from that in the characters, but the mean grey level and the variance may be fairly constant. The eye can quite easily distinguish between regions having the same mean grey level if one has a much higher variance than the other, providing the variation is within some kind of visual acuity. Similarly, other kinds of correlation between pixel values may appear as a `texture' variation

in the image, and this information may be extracted and put to use in obtaining a `feature' of an image. In the exercise at the end of chapter one where you were asked to distinguish pictures of underclad ladies from pictures of trees, even the most glorious of New England Fall Maple trees can easily be distinguished automatically from naked flesh, not by colour but by texture. The variation in a square block of 25 pixels is bigger for leaves than for flesh; try it if you don't believe me. I have studied such images with assiduous attention to detail and am confident of my ground.

Next Up Previous Contents

Next: Colour Images **Up:** Greyscale Images in general **Previous:** Quantisation *Mike Alder*
9/19/1997

Colour Images

- Generalities
- Quantisation
- Edge detection
- Markov Random Fields
- Measurements

Mike Alder
9/19/1997

Generalities

A colour image is normally three greyscale images. This may indeed be how the colour image was obtained: some of the astronomical pictures of planets and satellites of the solar system seen from spacecraft were obtained by putting a green filter and taking a greyscale image, then repeating with red and blue filters.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Edge detection **Up:** Colour Images **Previous:** Generalities

Quantisation

Quantising the graph of an image from $\mathbb{R}^2$ to $\mathbb{R}^3$ is only a little more difficult than quantising the graph of an image from $\mathbb{R}^2$ to $\mathbb{R}$, but the space in which the points sit is now five-dimensional. This ought not to worry those heroic enough to be reading this book. We are going to be looking for clusters in this space, and how this may be done will be described later, for we need to be very interested in clustering methods for other reasons.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Markov Random Fields **Up:** Colour Images **Previous:** Quantisation

Edge detection

If an edge is going to be really there and not some accident of brightness, then it ought to have some extension in space and also in colour, that is to say, if we think we have a sharp change in colour at some point, then neighbouring pixels ought to show similar changes. This means that it suffices to ensure that a putative edge in the red image ought to have spatial extension, likewise the green and blue images. So edge detection is slightly easier, because there are often simultaneous changes in more than one of the greyscale images at the same pixel. It would not be surprising if there were changes in the different greyscale images at nearby pixels, and smarter methods can look for this effect. Again, this sort of issue belongs in Image Processing courses, but it is worth doing a little meditation on what is possible before reading the books or attending the courses.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Measurements **Up:** Colour Images **Previous:** Edge detection

Markov Random Fields

As with greyscale images, texture effects may be described as local correlations between pixel values. Various models of this kind of phenomenon have been studied, including the theory of markov Fields. The work of Geman and Geman is notable in this regard: these are two people, not clones. See the bibliography for more details; I shall have a little more to say on this topic

later but it represents a degree of technicality inappropriate for the present book.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Spot counting **Up:** Colour Images **Previous:** Markov Random Fields

Measurements

If we can actually identify an object by some segmentation process involving boundaries, either of brightness or colour or texture, then again we can use mask methods or moments to try to obtain information about the shape of the object in the three colours. This gives us three separate sets of mask values or moments, and they may correlate. Indeed it is to be hoped that they do.

Finding objects out of correlations or textures alone is at the present time a convenience and tends to be image specific, as with the girly pix, rather than a pressing need. This may change

as the objects to be recognised get more complicated and more interesting.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: IR and acoustic Images **Up:** Image Measurements **Previous:** Measurements

Spot counting

One of the more fundamental things to be done in pattern recognition is simply counting the objects in an image. Going back to *Fig.2.10*, the paper clips, we see that the objects may overlap, and that this can very effectively clobber segmentation methods. If the objects are simple in shape, say disk shaped, then something can be done, but it is mostly nasty and *ad hoc* and therefore gives a thrill only to debauched computer programmers.

Some of the images on disk are microphotographs of cells, and it is frequently necessary to count the cells of a given type or colour. They usually overlap. Erosion of the cells until they fall into separate points, boundary smoothing and colour quantisation have been used by Dr. Chris deSilva and his students at the Centre for Intelligent Information Processing at the University of Western Australia, to obtain reliable cell counts. More advanced methods will be discussed later. It should be noted that there are an awful lot of cells to count, and automating the counting process is a critical bottleneck of mass screening schemes, in, for example, the early detection of breast cancer.

Next Up Previous Contents

Next: IR and acoustic Images **Up:** Image Measurements **Previous:** Measurements *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Quasi-Images **Up:** Image Measurements **Previous:** Spot counting

IR and acoustic Images

Not all images are produced by visible light, and images produced by acoustic or Infra-Red means are common, as are X-Rays and Tomographic images. Because there are a lot of people in this world who want to live forever (despite having no better idea of how to spend a weekend than watching football on television), a great deal of money is being poured into medical research, and some of this goes into automating the task of looking at images of people's insides. Since cutting them up in order to take visual images is considered intrusive and has side effects, anything which can give an image of a patients insides and allows him to walk away afterwards is a **Good Thing**.

Making sense of the resulting image, and correlating it to what you would have got if you'd opened the patient up with a hack-saw, is a task too complicated to be left to the medical profession, and hence the interest in PET scans, CAT scans, MRI scans, Acoustic Imaging, and generally the results

of pumping into patients anything too small to leave visible scars.

The military have what might be termed the converse interest in other than visual images; being able

to kill people and blow up buildings on a dark night is of obvious importance if you get your kicks that way. So being able to tell a tank from a Lamborghini by inspection of an Infra-Red image has its uses. If instead of being a military analyst safe in a bunker you are a missile trying to make up your mind whether or not to destroy yourself and your target, the automation issue arises.

Existing methods of analysis are essentially of the sort described for other images, but the source sometimes means that the funding situation is improved. It is for this reason that some energy and time has been dedicated to this class of data. Since many images are structured, syntactic methods have particular applicability.

Flaw detection in pipes, rails and machine parts by ultrasonic imaging, has been accomplished successfully using the methods I shall discuss in the chapter on Syntactic Pattern Recognition.

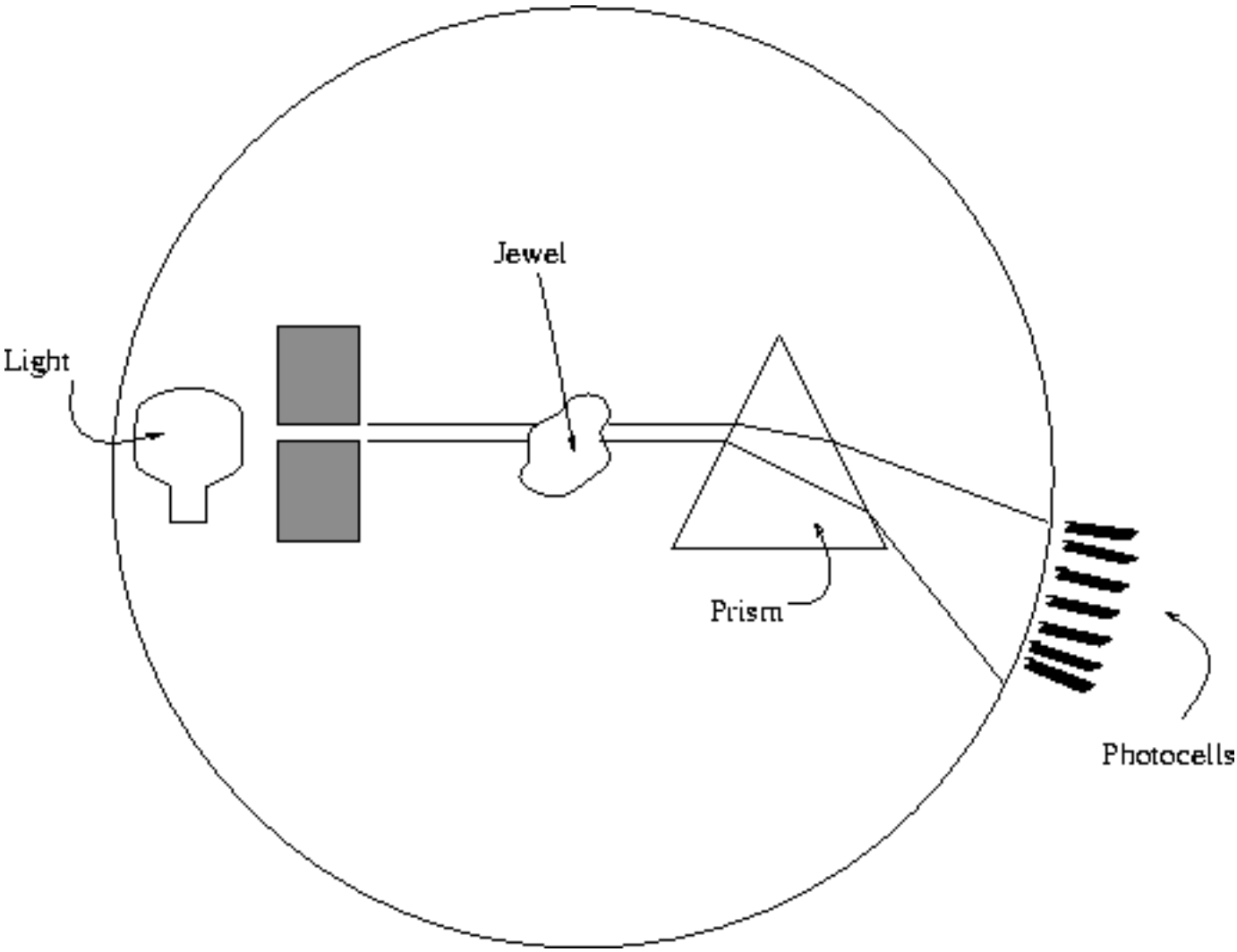
Next Up Previous Contents

Next: Quasi-Images **Up:** Image Measurements **Previous:** Spot counting *Mike Alder*
9/19/1997

Quasi-Images

In the case of a binary image, we are dealing with a pixel array in two dimensions, which it is convenient to describe as a point set in $\mathbb{R}^2$. There are many measurement processes which give point sets in higher dimensional spaces. For example, a standard method of sorting precious stones by colour is to place them on a tray and shine a light through the stone, as in *Fig.2.13*. The light is refracted to some extent by the stone, and the emergent beam is coloured. By placing a prism in the path of the emergent beam and an array of photocells across the spread out beam, the energy at different frequencies may be measured. If there are twelve such photoreceptors, the stone is transformed by the measuring process to a point in $\mathbb{R}^{12}$. The usual methods of cluster analysis and recognition may then be applied.

Figure 2.13: Measuring coloured glass.



Mike Alder
9/19/1997

Next Up Previous Contents

Next: Summary of Chapter Two **Up:** Image Measurements **Previous:** Quasi-Images

Dynamic Images

A video-recording of a football match shows figures moving around in ways which are, for short periods, relatively predictable. Using temporal information to identify objects in a time series of images is possible using such techniques as differencing consecutive images to distinguish the moving objects from the background. It is, of course, very simple to difference consecutive images and simply measure the amount of change. A simple intruder alarm which accomplishes this is currently obtainable for under fifty dollars. Unfortunately it counts newspapers blown by the wind and passing cars as intruders. Actually being able to distinguish between a man, a sheet of newspaper, and a passing car is possible by the techniques described. More sophisticated recognition issues, such as distinguishing a man from a dog, naturally arise. Because of the huge variety of images which a human being can confidently label as `man' or `dog', the existing methods are not satisfactory. It is possible to submit the whole image to a huge neural net, and this has been tried by optimists in need of an education, but any passing statistician would give a pitying smile if informed of this plan. It is something like trying to recognise text a page at a time because it is hard to segment into letters. You'd have to be pretty desperate to try it.

Next Up Previous Contents

Next: Summary of Chapter Two **Up:** Image Measurements **Previous:** Quasi-Images *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Exercises **Up:** Image Measurements **Previous:** Dynamic Images

Summary of Chapter Two

This chapter has discussed, in outline, material which can be found in detail in books on Image Processing; references to such books may be found in the bibliography following. It has, I hope, put the material into some kind of perspective at the expense of being sketchy and superficial. The exercises, if done diligently, will also fill in some details which have been left out.

The usual problem when confronted with an image is that it contains rather a lot of things not just one. This leads to the segmentation problem; trying to chop up the image so each part of it contains just one recognisable object. Having isolated the parts, it is then possible to move on to the task of identifying them. It is irritating in the extreme that human beings seem to do this in the reverse order.

Segmentation will often entail boundary tracing which in turn involves edge detection for greyscale images. This may be conveniently accomplished by convolution filters. I have sketched out the underlying idea here, but to find out the details, more specific texts are indicated.

Having segmented the image, which in easy cases may be done by finding clear spaces between the objects and in hard cases cannot be done at all, we may throw away the bits that cannot correspond to objects we can recognise. We may then choose to normalise the result of this; this might be a matter of inserting the object in a standard array, or moving it and scaling it so that it just fits into the unit disk.

We then have two basic methods of turning the image into a vector; one is to use a `mask' family which may inspect bits of the image and compare the fit to some standard mask value. This does a `local' shape analysis of some sort. The alternative is a moment based method, which includes FFTs, DCTs and Zernike moments as special cases of orthogonal basis functions.

Images may be obtained from a wide variety of sources, and simply counting the objects in an image may present formidable problems.

Next Up Previous Contents

Next: Exercises **Up:** Image Measurements **Previous:** Dynamic Images *Mike Alder*
9/19/1997

Exercises

1. Generate a square grid such as that in the background of *Fig.2.6*, if necessary using pen and paper, and mark in some pixels. Repeat with another grid and mark the origin in both grids. Now compute carefully the dilation and erosion of one by the other. Write a program to dilate and or erode one pixel array by another and display all three arrays.
2. With the program of the last exercise, explore alternate erosions and dilations on some .tif files obtained from a scanner or from disk.
3. Use erosion techniques to count the objects in *Fig.2.10* or *Fig.2.12.*
4. Write a program to find the lines of text and the spaces between them in some scanned image of text such as *Fig.2.4.* You may use any method you choose.
5. Extend the previous program so as to separate out each line of text into words.
6. Extend the previous program so as to isolate single characters. What does it do to an image such as *Fig.2.2*?
7. Write a boundary tracing program and try it out on a .tif file of your choice. Alternatively, you might like to start on a bitmap file if you have access to a unix workstation and some familiarity with the beast. See the *.c* files on disk for a helping hand.
8. By hook or by crook put a box around each of some characters in a printed word, either from *Fig.2.4* or somewhere else. Normalise the boxes to a standard size, having first worked out what a good standard size is. Print your normalised images and ask whether you could recognise them by eye! Recall that a 2 by 2 matrix applied to each pixel will give the effect of a linear transformation. It is recommended that you expand or shrink each axis separately to a standard size as your first move.
9. How many scan lines would you think necessary to distinguish the characters in your sample?

   Turn each character into a vector by scanline methods. Can you tell by looking at the vectors what

the characters are? (It might be useful to start of with a set of digits only, and then to investigate to see how things change a the alphabet size increases.)

10.

Are there any alternative mask shapes you feel tempted to use in order to convert each character into a vector?

11.

Segment by hand an image such as *Fig.2.12* and normalise to occupy the unit disk with the centroid of the function representing the region at the origin. Divide the disk up into sectors and some the quantity of grey in each sector. Can the resulting vectors distinguish shapes by eye? Can you devise any modifications of this simple scheme to improve the recognition?

12.

Obtain images of shapes which have some kind of structure which is complicated; orient them in some canonical way and look at them hard. Can you devise a suitable collection of masks to discriminate the shapes? Characters under grey levels present an interesting problem set. Try getting camera images of stamped digits (easily made with modelling clay and a set of house numbers) under different illuminations.

13.

Compute the central moments of one of your characters up to order two by hand. Write a program for doing it automatically and confirm it gives the right answer. Extend your program to compute central moments up to any order. Compute them up to order n for all your characters, and check by eye to see if you can tell them apart by looking at the vectors, for different values of n.

14.

Write a program to normalise any array of characters into the unit disk with the centroid as origin.

15.

Write a program to compute the first 12 Zernike moments for a pixel array which has been normalised into the unit disk.

16.

Use a border following algorithm to extract external borders of your characters and then apply the last program to see if you can still recognise the characters by looking at the vectors obtained from taking moments of the boundaries.

17.

Find an algorithm for finding edges in greyscale images and use it to differentiate an image, bringing the borders of objects into relief. The disk programs might be places to start looking, but you may find other books of value.

18.

Write a program which smooths a time series of real numbers by applying a moving average filter. The program should ask for the filter coefficients from -n to n after finding out what n is. To be more explicit, let $x(n)$ be a time series of real numbers; to get a good one, take the daily exchange rate for the Dollar in terms of the Yen for the last few weeks from the past issues of any good newspaper. Let $g(n)$ be defined to be zero for $|n| > 5$ and assign positive values summing to 1 for

$-5 \leq n \leq 5$. You could make them all equal, or have a hump in the middle, or try some negative values if you feel brave.

Your program should generate a new time series $y(n)$ defined by

$$y(n) = \sum_{j=-\infty}^{j=\infty} x(n-t)g(t)$$

and plot both x and y.

19.

Modify your program to deal with a two dimensional filter and experiment with it; get it to remove `salt and pepper' noise introduced into a grey scale image by hand.

20.

Beef up the above program even further, and use it for differentiating images. Try to segment images like *Fig.2.12* by these means.

21.

Generate a binary waveform $x(n)$ as follows: if the preceding value is black, make the current value white, and if the preceding value is white, make the current one black. Initialise at random. This gives a trivial alternating sequence, so scrap that one and go back the preceding two values.If the two preceding values are both black, make the current value white; if the last two values were black then white, make the current value white; if the last two values were white make the current value black, and if the last two values were white then black, make the current value black. Well, this is still pretty trivial, so to jazz it up, make it probabilistic. Given the two previous values, make a random choice of integer between 0 and 9. If both the preceding values were black, and the integer is less than 8 make the current value white, if it is 8 or 9 make it black. Make up similar rules for the other three cases. Now run the sequence from some initial value and see what you get.

22.

Could you, given a sequence and knowing that it was generated according to probabilities

based on the preceding k values as in the last problem, work out what the probabilities are?

23.

Instead of having k = 2, does it get more or less interesting if k is increased?

24.

Instead of generating a sequence of binary values, generate a sequence of grey scale values between 0 and 1 by the same idea.

25.

Can you generalise this to the case of two dimensional functions? On a rectangular pixel array, make up some rules which fix the value of a pixel at 0 or 1 depending on the values of pixels in the three cells to the North, West and North-West. Put a border around the array and see if you can find rules for generating a chess-board pattern. Make it probabilistic and see if you can generate an approximate texture. Do it with grey levels instead of just black and white.

26.

Make up a probabilistic rule which decides what a cell is going to have as its value given all the neighbouring pixel values. Generate a pixel array at random. Now mutate this initial array by putting a three by three mask on, and using the surrounding cells to recompute the middle one. Put your mask down at random repeatedly. Do the following: always make the centre pixel the average of the surrounding pixels. Now make the top and left edge black, the bottom and right edge white, and randomise the original array. Can you see what must happen if you repeat the rewrite operation indefinitely? Run this case on your program if you can't see it.

27.

Can you, in the last set of problems, go back and infer the probabilistic rules from looking at the cell values? Suppose one part of an array was obtained by using one set of values and another by using a different set, can you tell? Can you find a way of segmenting regions by texture?

28.

Take an image of some gravel on white paper, then sieve the mixture and take another image of what gets through the sieve and another of what gets left in the sieve. Repeat six times. Can you sort the resulting images by any kind of texture analysis? Try both high and low angle illumination in collecting your data.

Next Up Previous Contents

Next: Bibliography **Up:** Image Measurements **Previous:** Summary of Chapter Two *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Statistical Ideas **Up:** Image Measurements **Previous:** Exercises

Bibliography

1. Terry Pratchett *Moving Pictures* Gollancz 1990.
2. Kreider, Kuller, Ostberg and Perkins, *An Introduction to Linear Analysis*, Addison Wesley 1966.
3. Wojciech Kuczborski, *Real Time Morphological Processes based on Redundant Number Representation*, Ph.D. thesis, The University of Western Australia, 1993.
4. Tony Haig, *Reconstruction and Manipulation of 3-D Biomedical Objects from Serial Tomographs*Ph.D. thesis, University of Western Australia 1989.
5. Tony Haig, Yianni Attikiouzel and Mike Alder, *Border Following: New Definition Gives Improved Border* IEE Proc part I, Vol 139 No 2 pp206-211 April 1992.
6. Jim R. Parker, *Practical Computer Vision using C* New York: Wiley, 1994.
7. Edward R. Dougherty and Charles R. Giardina, *Image Processing Continuous to Discrete*, *Vol.1*. Prentice-Hall 1987.
8. Edward R. Dougherty and Charles R. Giardina, *Matrix Structured Image Processing* Prentice-Hall 1987.
9. Charles R. Giardina and Edward R. Dougherty, *Morphological methods in image and signal processing* Prentice-Hall, 1988.
10. Robert M. Haralick and Linda G. Shapiro, *Computer and Robot Vision* Addison-Wesley 1991
11. IEEE Transactions on Image Processing : a publication of the IEEE Signal Processing Society, from: Vol. 1, no. 1 (Jan. 1992)
12. John C. Russ, *The image processing handbook* CRC Press, 1992.

13.

Theo Pavlidis, *Algorithms for graphics and image processing* Berlin Springer-Verlag, 1982.

14.

Azriel Rosenfeld,(Ed) *Image modeling* New York : Academic Press, 1981.

15.

Azriel Rosenfeld, Avinash C. Kak. *Digital picture processing* New York Academic Press, 1982.

16.

Laveen N. Kanal and Azriel Rosenfeld,(Eds) *Progress in pattern recognition* Elsevier North-Holland, 1981

17.

Jacob Beck, Barbara Hope, Azriel Rosenfeld, (Eds) *Human and machine vision* New York Academic Press, 1983.

18.

Azriel Rosenfeld, editor *Human and machine vision II* Boston : Academic Press, 1986.

19.

Rosenfeld, Azriel, *Picture processing by computer* New York : Academic Press, 1969.

20.

Julius T. Tou and Rafael C. Gonzalez, *Pattern recognition principles* Addison-Wesley Pub. Co., 1974

21.

Rafael C. Gonzalez, Richard E. Woods, *Digital image processing 3rd ed* Addison-Wesley, 1992.

22.

Serra, Jean Paul, *Image analysis and mathematical morphology*. New York : Academic Press, 1982.

23.

Shimon Ullman and Whitman Richards, (Eds) *Image understanding* Norwood, N.J. : Ablex Pub. Corp., 1984.

24.

Narendra Ahuja, Bruce J. Schachter, *Pattern models* New York : Wiley, 1983 .

25.

Wang, Patrick Shen-pei, (Ed) *Computer vision, image processing and communications systems and applications : proceedings of the 1985 NEACP Conference on Science and Technology, Boston, USA, Nov. 30-Dec. 1, 1985* Philadelphia : World Scientific, 1986.

26.

Wang, Patrick Shen-pei, (Ed)*Array grammars, patterns and recognizers* World Scientific, 1989.

27.

K. S. Fu and A. B. Whinston, (Eds) *Pattern recognition theory and applications [proceedings of the NATO Advanced Study Institute on Pattern Recognition-Theory and Application, Bandol,*

France, September 1975] Leyden Noordhoff, 1977.

28.

K.S. Fu, (Ed) *Applications of pattern recognition* CRC Press, 1982.

29.

King-Sun Fu and T.L. Kunii, (Eds) *Picture engineering* Springer-Verlag, 1982.

30.

K.S. Fu , with contributions by T.M. Cover, *Digital pattern recognition* Springer-Verlag, 1980.

31.

Tzay Y. Young and King-Sun Fu, *Handbook of pattern recognition and image processing* Academic Press, 1986.

32.

George Robert Cross *Markov random field texture models [microform]* Ann Arbor, Mi. University Microfilms, 1981, 3 microfiche (Ph.D. thesis)

33.

Alireza Khotanzad and Jiin-Her Lu, *Classification of Invariant Image Representations Using a Neural Network.* IEEE Trans. Speech and Sig.Proc. Vol.38 No.6, June 1990.

34.

F.Zernike, *Beugungstheorie des Schneidenverfahrens und Seiner Verbesserten Form, der Phasenknotrastemethode* Physica Vol.1.pp 689-704, 1934.

Next Up Previous Contents

Next: Statistical Ideas **Up:** Image Measurements **Previous:** Exercises *Mike Alder*
9/19/1997

Statistical Ideas

In chapter one I explained that there were two parts to Pattern Recognition; the first was finding a suitable system of *measurement*, so that each object got turned into a vector, a point in . The second part consisted of comparing the results of doing this on a new object, with some set of data where we know the category to which the object belongs, thus comparing vectors with vectors.

The second chapter looked at ways of measuring objects- mostly pixel arrays, since these arise from pointing cameras at the world. In this chapter and the next, I shall start on the task of doing the actual recognition using statistical methods. In the following chapter I shall discuss neural nets, and in later chapters I shall treat syntactic methods.

We assume then that the robot has sensed the world, maybe by pointing a camera at it, possibly by other means, and that some measuring process has been applied to obtain a vector of real numbers. The robot has seen other such vectors in the past and has also been told what category of object they describe: now it has to make up its own mind about the latest vector. An industrial robot looking at something to be sorted, or possibly welded, may easily be in this situation.

I find myself, yet again, in a difficult position. It is not hard to give recipes which are easy to implement in programs, and to pass lightly over the issues concerning why they work, or indeed if they work. This goes against the grain; it is necessary to give a survey of what is standard practice, but it is also necessary to look at standard practice with a cold, critical and careful eye if there is to be any progress. The situation is particularly acute in Statistics. Statistics is often taught as recipes, with the deplorable consequences I have mentioned in chapter one, and it is about time this repellent habit was abandoned.

On the other hand, sorting out fundamental ideas in Probability theory carefully is a very technical business indeed, and many issues are still in dispute. So this chapter starts off somewhere between Scylla and Charybdis, trying to get fundamental ideas examined in informal language. This may show more courage than judgement; you can decide whether or not you feel illuminated.

I shall in this chapter, then, go into foundational issues with more enthusiasm than is common in Pattern Recognition texts. When we come to investigate Syntactic Pattern Recognition we shall come up against conceptual difficulties related to the kinds of models of neural processing which these methods imply. We might as well start the way we intend to continue and clear away the undergrowth from the beginning. In the spirit of critical reflection which I propose to stimulate, I shall draw attention to the underlying assumptions and conceptual underpinnings of contemporary practice. It is hard to change your own ideas, and even harder if you don't know what they are. Those who feel uncomfortable with theoretical issues should appreciate that there is nothing so practical as a good theory, and nothing so dangerous as a bad one.

The present chapter then is about the general ideas of statistics and probability theory which have a

bearing on pattern recognition; the next chapter will give the recipes. I have broken the material up in this way because painful experience has compelled me to recognise that many people who claim to be of a practical disposition object to generalities and find ideas confusing. `Don't give me all that blather, just give me the formulae', they say. Thought hurts their heads and gives them doubts about their certainties, so they want nothing to do with it. I have taken pity on them; this chapter therefore, may be skipped by those for whom thinking is an unnatural act. They may, of course, regret the decision later.

- History, and Deep Philosophical Stuff
  - The Origins of Probability: random variables
  - Histograms and Probability Density Functions
  - Models and Probabilistic Models
- Probabilistic Models as Data Compression Schemes
  - Models and Data: Some models are better than others
- Maximum Likelihood Models
  - Where do Models come from?
- Bayesian Methods
  - Bayes' Theorem
  - Bayesian Statistics
  - Subjective Bayesians
- Minimum Description Length Models
  - Codes: Information theoretic preliminaries
  - Compression for coin models
  - Compression for *pdf*s
  - Summary of Rissanen Complexity
- Summary of the chapter
- Exercises
- Bibliography

Next Up Previous Contents

Next: History, and Deep Philosophical **Up:** An Introduction to Pattern **Previous:** Bibliography *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: The Origins of Probability: **Up:** Statistical Ideas **Previous:** Statistical Ideas

History, and Deep Philosophical Stuff

- The Origins of Probability: random variables
- Histograms and Probability Density Functions
- Models and Probabilistic Models

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Histograms and Probability Density **Up:** History, and Deep Philosophical **Previous:** History, and Deep Philosophical

The Origins of Probability: random variables

Probability Theory originated in attempts to answer the practical questions of whether it is better to discard one card to fill a straight or two to fill a flush, and similar questions regarding dice. No branch of Applied Mathematics has better credentials as a practical and necessary part of a gentleman's education. If the theory worked, the customer made a buck, and if it didn't he tried to get his losses back from the theoretician; seldom has the motivation for honest theorising been so pointed.

The problem referred to arises in draw poker where you are one of some number of players, say four for definiteness, who have each been dealt five cards. You look at your hand and see, let us suppose, ♡4, 5, 6; ♣7; ♠Q. You are allowed to discard one or two cards and get replacements off the top of what is left of the pack. The aim is, as all but the youngest are aware, to get a `good' hand. Two possibilities immediately suggest themselves: to discard the Queen of Spades and hope to fill the straight by picking either a 3 or an 8; or to discard the club and spade and draw two cards in the hope of filling the flush by getting two more hearts.

The argument proceeds as follows: There are 47 cards not in my hand, and we know nothing about them beyond the plausible assumption that they are from a standard pack, and in a random order- whatever that means. In which case the 10 hearts left are somewhere in those 47 cards you haven't got. So there is a probability of 10/47 that the next card on the stack is a heart, and of 9/46 that the card after that is also a heart. So the probability of getting your flush is 90/2162, approximately 0.0416. If you aim for the straight, there are eight possible cards which can do the job, a 3 or 8 of any suit will do, and there are 47 left, so the probability of the top card being acceptable is 8/47 or approximately 0.1702. In rough terms, its about a one in six shot that you'll fill the straight, and only about one in twenty four that you'll fill the flush.

Which suggests you should go for the straight, but a flush beats a straight, so maybe you should go for the flush and improve your chances of winning more money. Who is to tell?

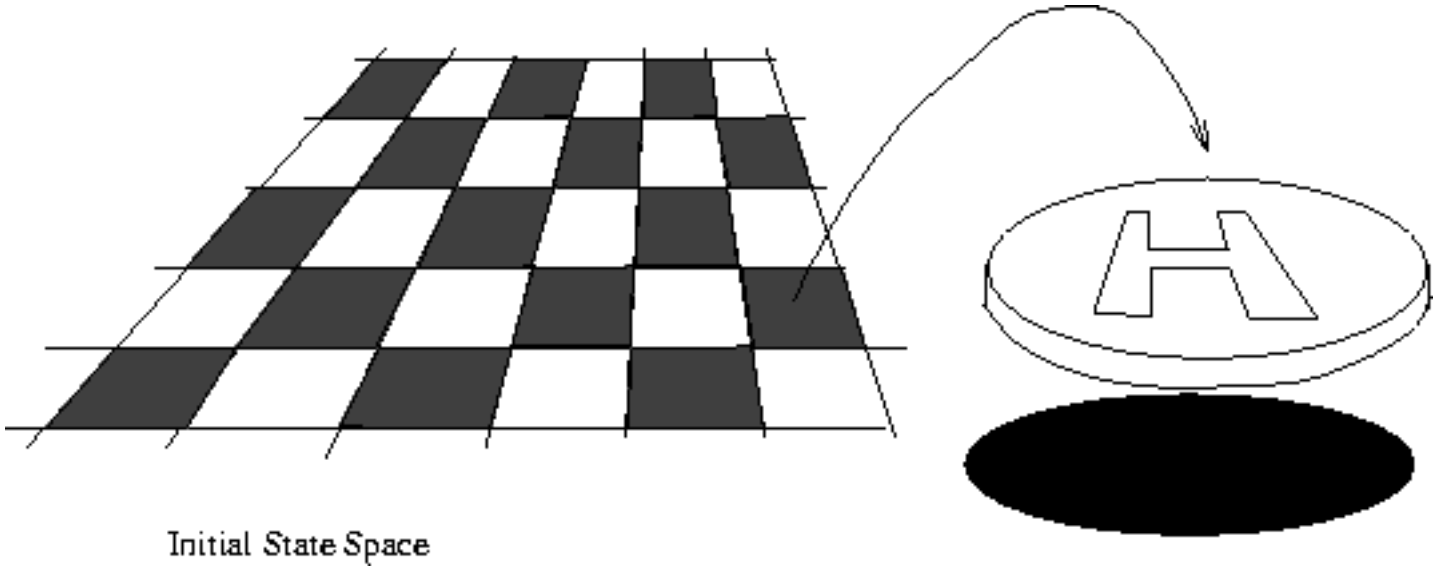
Now the analysis is very standard but contains a lot of assumptions, such as that the deck is not rigged and the other players will not shoot you under the table if you win. As everybody who has played poker knows, there may be more useful information in the momentary smug look on the face opposite than in the above calculation. The assumptions are not usually much thought about, which is why, perhaps, probabilists are not noticeably richer than the rest of us.

The great Australian contribution to gambling is the game of *two-up*, which consists of throwing two coins in the air. If they land with the same face showing, two heads or two tails, then one party wins, otherwise he doesn't. While they are in the air, an attempt is made to affect the outcome by shouting such advice as `*Come in Spinner*' at the coins. Contrary to what one might expect, this works. But only about

half the time.

Now there is nothing much random about the next two cards on the pack, either they are both hearts or they aren't, and likewise it is rather strange that throwing coins in the air should have anything random about it, since Newton's laws apply and the initial state determines the final state completely. The fact is, we don't have much useful information about the initial state, and the dynamical calculations on bouncing are somewhat beyond most players. Persi Diaconis, a statistician who started academic life somewhat non-canonically (much to the joy of his students and most of his colleagues), can reliably toss a coin so as to produce whatever outcome he wants. Score one for Isaac and Persi. But most of us will take it that initial states which are, to us, indistinguishable, can and do produce different outcomes.

Figure 3.1: A random variable.



Now suppose we take the space of initial positions, orientations, momenta and angular momenta for the coin and assign, in accordance with Newton's laws, a colour to each point in this space, black if the coin comes down heads and white if it comes down tails. Then the twelve dimensional space of initial states is divided up into a sort of checker-board, alternate black and white regions, separated by rather thin bits corresponding to the cases where the coin finishes on its edge. We cannot say much about the actual initial state, because the regions of black and white are so small that we see only a grey blur; but symmetry arguments lead us to the belief that there is as much black as there is white. The hypervolume or *measure* of the black points is equal to the measure of the white points. If we take the total measure to be 1, for convenience only, then we can assign the measure of the Heads outcome as being pretty much 1/2- assuming, in the face of the evidence, that the coin is truly symmetric. We summarise this by saying that if the coin is fair then the probability of getting heads is 0.5. And what this means is that the measure of all the regions in which the coin starts its throw which will lead by the inexorable action of natural law to its coming up Heads, is one half.

Intuitively, in the case of a finite set, the measure of the set is proportional to the number of elements in it; in the case of a subset of the real line, the measure of a set is the length of the set; in two dimensions it

is the set's area, in three its volume. It is possible to abstract a collection of properties that all these things have in common, and then we list these properties and call them the axioms for a *measure space*. Then anything having these properties is an example, including the ones we started with. This is exactly like taking out the common properties of $\mathbb{R}^n$ and calling them the axioms for a Vector Space; it is standard Mathematical practice and allows us to focus our minds on the essentials. We thereupon forget about the dimension of any space of initial states, and indeed almost all properties of it, except the fact that it has some collection of subsets each of which has a measure, and that it has total measure 1. The unit square in the plane is simply a particular example of such a space.

It is thought along these lines which leads us to the idea of a *random variable* (*rv*).

We imagine a map from some space (of initial conditions?) to a space of outcomes, and we suppose that nothing is known about the domain of the map except the measure of the set which produces each type of outcome. At least, this is what we assume it means if there are only finitely many outcomes; if there is a continuum of outcomes, as when someone throws a dart at a board, we suppose that there is a measure for any measurable set in which the dart might land. By putting sensible conditions on what a measure ought to be, making it behave in the same way as area, volume, *et cetera*, and similar conditions on how maps ought to treat measures, we come out with a definition of a random variable as a map from some inscrutable space having a measure and nothing much else, into $\mathbb{R}^n$ for some n, sometimes with restrictions such as taking values only in 0,1 with $n = 1$. Now we can define an *event* as a set of possible outcomes, and the *probability of an event A, p(A)*, to be the measure of all those points in the domain of the *rv* which lead to us getting an outcome in A. This is why probability behaves mysteriously like area of sets, and why those little diagrams of circles intersecting, which figure in the elementary texts on Probability Theory, actually work.

In short we imagine that any random variable is much like throwing a die or a dart. This imagery keeps probabilists going long after the logic has run dry.

Similarly, your ignorance of the state of the stack of cards from which you will be dealt the next one or two you ask for, is, we shall assume, close to total. Of all the possible arrangements in the stack of cards left, you deem them equally likely, by virtue of your innocent faith in the shuffling process to which the pack was subjected. To obtain the given `probabilities' by the arithmetic process is a trivial combinatorial argument which most people find convincing, both as providing some relative degree of belief in the two outcomes, and also an estimate of what might happen if you were in this situation a large number of times and kept count of the results.

Experiments have been done on packs of cards to see if the results are what the arguments predict. Some anomalies have been reported, but mainly by Psychic investigators with an axe to grind. It is within the capacity of all of us to toss a coin a hundred times. (Persi Diaconis has done it much more often!) I suggest you try the experiment and see how persuasive you find the argument in the light of the experimental data.

Next Up Previous Contents

Next: Histograms and Probability Density **Up:** History, and Deep Philosophical **Previous:** History, and Deep Philosophical *Mike Alder*

9/19/1997

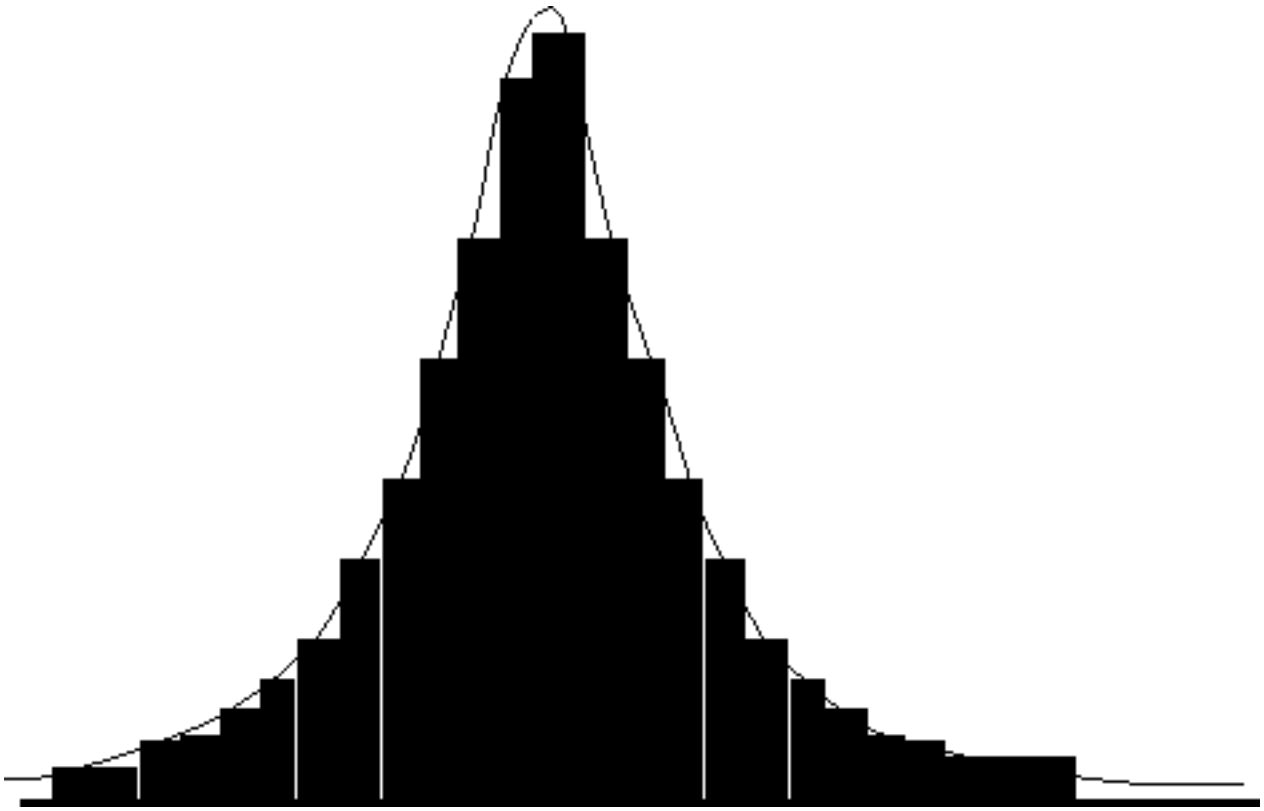
Next Up Previous Contents

Next: Models and Probabilistic Models **Up:** History, and Deep Philosophical **Previous:** The Origins of Probability:

Histograms and Probability Density Functions

If there are only a finite number of possible things we might have as outcomes, we can enumerate them, 1 to *k*, and, if the random variable is really given to us explicitly, compute the measure for each of them. We could draw a histogram of outputs and *probabilities* associated with them. Such a thing is called a *probability distribution* and we are all familiar with them. For a coin which can only come down Heads or Tails, we can let 0 represent Tails, 1 Heads, and erect a little pole of height 0.5 over each of these two numbers.

Figure 3.2: Histogram/PDF for a *RV* taking values in $\mathbb{R}$.



If the random variable takes a continuum of values, as when someone hurls darts at a board, we cannot assign a probability to any single outcome, since under normal circumstances this will be zero.

But we can still draw a histogram for any choice of boxes partitioning the space, $\mathbb{R}^n$, of values of the *rv*. If we normalise, so that the area, volume or in general measure under the histogram is one, and then

do it again with a finer partition, we can get closer to a continuous distribution. In the limit, with some technical conditions satisfied that you can safely ignore because they are no more than mathematical book-keeping, you may get a continuous non-negative function over , with integral 1, and this is known as a *probability density function*, *pdf* for short, and again they are exceedingly familiar. There are some niceties; the *pdf* may not be continuous and the values may be mixed discrete and continuous, but we need not contemplate these issues in our applications. The usual derivation of the *pdf* from the measure is different from the present hint that you do it by limits of histograms, and has little to recommend it unless you are an analyst.

It is worth noting that since we can only measure vectors to some finite precision and we absolutely never get a data set which is of uncountably infinite cardinality, the distinction between a very fine histogram and a real *pdf* is somewhat metaphysical. It is also worth noting that a hypothetical measure space and map from it, about which nothing can be said except the relative meaures of bits corresponding to sets of outputs from the map, is also a touch metaphysical. If you can use one model for the *rv* and I can use another with a different domain space, and if we agree on every calculation about what to expect, then it must cross one's mind that we might manage to do very well without having an *rv* at all. All we really need is a measure on the space of outcomes, the *sample space*. If we reflect that to specify a measure on a sample space which is some region in the simplest way is by giving a density function, we see that it might have been simpler to start with the *pdf*.

Next Up Previous Contents

Next: Models and Probabilistic Models **Up:** History, and Deep Philosophical **Previous:** The Origins of Probability: *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Probabilistic Models as Data **Up:** History, and Deep Philosophical **Previous:** Histograms and Probability Density

Models and Probabilistic Models

Of course, nobody comes along and gives you a random variable. What they do usually is to give you either a description of a physical situation or some data and leave it to you to model the description by means of an *rv*. The cases of the cards and the coins are examples of this. With the coins, for example, you may take the 12-dimensional state space of physics if you wish, but it suffices to have a two point space with measure 0.5 on each point, the map sending one to Heads and the other to Tails. For *two-up*, you can make do with a two point space, one labelled `same', the other labelled `different', or you can have two copies of the space of Heads and Tails with four points in it, a pair of them labelled `same', the other pair `different', or a twenty-four dimensional state space with half the space black and the other half white- it doesn't much matter.

The histograms over the space of outcomes are the same.

Statisticians sometimes say that the *rv*, or the histogram or *pdf*, is a *model* for the actual data. In using this terminology they are appealing to classical experience of mathematical models such as Newtonian Dynamics. There are some differences: in a classical mathematical model the crucial symbols have an interpretation in terms of measurables and there are well defined operations, such as weighing, which make the interpretation precise. In the case of probabilistic models, we can measure values of a physical variable, but the underlying mechanism of production is permanently hidden from us. Statistical or probabilistic models are not actually much like classical mathematical models at all, and a certain kind of confidence trick is being perpetrated by using the same terminology. Let us look at the differences.

Newton modelled the solar system as a collection of point masses called planets revolving around another point mass called the sun, which was fixed in space. The reduction of a thing as big as a planet to a single point had to be carefully proved to be defensible, because we live on one of them and have prejudices about the matter. The Earth doesn't look much like a point to us. It can be shown that the considerable simplification will have no effect on the motion provided the planets are small enough not to bump into each other, and are spheres having a density function which is radial. Now this is not in fact true. Planets and suns are oblate, and the Earth is not precisely symmetric. But the complications this makes are not large, so we usually forget about them and get on with predicting where the planets will be at some time in the future. The results are incredibly good, good enough to send space craft roaring off to their destiny a thousand million miles away and have them arrive on schedule at the right location. Anybody who doesn't thrill to this demonstration of the power of the human mind has rocks in his head. Note that the model is not believed to be *true*, it is a symbolic description of part of the universe, and it has had simplifications introduced. Also, we don't have infinite precision in our knowledge of the initial state. So if the spacecraft is out by a few metres when it gets to Saturn, we don't feel too bad about it. Anybody who could throw darts with that precision could put one up a gnat's bottom from ten kilometres.

If we run any mathematical model (these days often a computer simulation) we can look at the measurements the model predicts. Then we go off and make those measurements, and look to see if the numbers agree. If they do, we congratulate ourselves: we have a good model. If they disagree but only by very small amounts that we cannot predict but can assign to sloppy measurement, we feel moderately pleased with ourselves. If they differ in significant ways, we go into deep depression and brood about the discrepancies until we improve the model. We don't have to believe that the model is *true* in order to use it, but it has to agree with what we measure when we measure it, or the thing is useless. Well, it might give us spiritual solace or an aesthetic buzz, but it doesn't actually do what models are supposed to do.

Probabilistic models do not generate numbers as a rule, and when they do they are usually the wrong ones. That is to say, the average behaviour may be the same as our data set, but the actual values are unlikely to be the same; even the model will predict that. Probabilistic models are models, not of the measured values, but of their distribution and density. It follows that if we have very few data, it is difficult to reject any model on the grounds that it doesn't fit the data. Indeed, the term `data' is misleading. There are two levels of `data', the first is the set of points in , and the second is the distribution and density of this set which may be described by a probabilistic model. Since we get the second from the first by counting, and counting occurences in little cells to get a histogram as often as not, if we have too few to make a respectable histogram we can't really be said to have any data at the level where the model is trying to do its job. And if you don't have a measurement, how can you test a theory? This view of things hasn't stopped people cheerfully doing hypothesis testing with the calm sense of moral superiority of the man who has passed all his exams without thinking and doesn't propose to start now.

Deciding whether a particular data set is plausibly accounted for by a particular probabilistic model is not a trivial matter therefore, and there are, as you will see later, several ways of justifying models. At this point, there is an element of subjectivity which makes the purist and the thoughtful person uneasy. Ultimately, any human decision or choice may have to be subjective; the choice of whether to use one method or another comes easily to the engineer, who sees life as being full of such decisions. But there is still an ultimate validation of his choice: if his boiler blows up or his bridge falls down, he goofed. If his program hangs on some kinds of input, he stuffed up. But the decision as to whether or not the probabilistic advice you got was sound or bad is not easily taken, and if you have to ask the probabilist how to measure his success, you should do it before you take his advice, not after.

A probabilistic model then does not *generate* data in the same way that a causal model does; what it can do, for any given measurement it claims to model, is to produce a number saying how likely such a measurement is. In the case of a discrete model, with finitely many outcomes say, the number is called the *probability* of the observation. In the case of a continuous *pdf* it is a little more complicated. The continuous pdf is, recall, a limit of histograms, and the probability of getting any specified value is zero. If we want to know the probability of getting values of the continuous variable within some prescribed interval, as when we want to know the probability of getting a dart within two centimetres of the target's centre, we have to take the limit of adding up the appropriate rectangular areas: in other words we have to integrate the *pdf* over the interval. For any outcome in the continuum, the *pdf* takes, however, some real value. I shall call this value the *likelihood* of the outcome or event, according to the model defined by the *pdf*.

If we have two data, then each may be assessed by the model and two probabilities or likelihoods output

(depending on whether the model is discrete or continuous); multiplying these numbers together gives the probability or likelihood of getting the pair on independent runs of the model. It is important to distinguish between a model of events in a space of observables applied twice, and a model where the observables are pairs. The probability of a pair will not in general be the product of the probabilities of the separate events. When it is, we say the events are *independent*.

For example, I might assert that any toss of a coin is an atomic phenomenon, in which case I am asserting that the probability of any one toss producing heads is the same as any other. I am telling you how to judge my model: if you found a strict alternation of heads and tails in a sequence of tosses, you might reasonably have some doubts about this model. Conversely, if I were to model the production of a sequence of letters of the alphabet by asserting that there is some probability of getting a letter `u' which depends upon what has occurred in the preceding two letters, the analysis of the sequence of letters and the inferences which might be drawn from some data as to the plausibility of the model would be a lot more complicated than the model where each letter is produced independently, as though a die were being thrown to generate each letter. Note that this way of looking at things supposes that probabilities are things that get assigned to events, things that happen, by a model. Some have taken the view that a model is a collection of sentences about the world, each of which may often contain a number or numbers between 0 and 1; others, notably John Maynard Keynes, have taken the view that the sentence doesn't contain a number, but its truth value lies between 0 and 1. The scope for muddle when trying to be lucid about the semantics of a subject is enormous.

Another difference between probabilistic models and causal models is the initial conditions. Most models are of the form: *if* condition A is observed or imposed, *then* state B will be observed. Here condition A and state B are specified by a set of measurements, i.e by the values of vectors obtained by prescribed methods of measurement. Now it is not at all uncommon for probabilistic models to assume that a system is `random'. In the poker calculation for example, all bets are off if the cards were dealt from a pack which had all the hearts at the top, for my having three hearts means that so do two of the other players and the last has four. So the probability of a flush being filled is zero. Now if you watched the dealer shuffle the cards, you may believe that such a contingency is unlikely, but it was the observation of shuffling that induced you to feel that way. If you'd seen the dealer carefully arranging all the hearts first, then the spades and clubs in a disorganised mess, and then the diamonds at the end, you might have complained. Why? Because he would have invalidated your model of the game. Now you probably have some loose notion of when a pack of cards has been randomised, but would you care to specify this in such a way that a robot could decide whether or not to use the model? If you can't, the inherent subjectivity of the process is grounds for being extremely unhappy, particularly to those of us in the automation business.

The terminology I have used, far from uncommon, rather suggests that some orders of cards in a pack are `random' while others are not, and shuffling is a procedure for obtaining one of the random orders. There are people who really believe this. Gregory Chaitin is perhaps the best known, but Solomonoff and Kolmogorov, also take seriously the idea that some orders are random and others are not. The catch is that they define `random' for sequences of cards as, in effect, `impossible to describe briefly, or more briefly than by giving a list of all the cards in order'. This makes the sequence of digits consisting of the decimal expansion of π from the ten thousandth place after the decimal point to the forty thousandth place very much non-random. But if you got them printed out on a piece of paper it would not be very practical to see the pattern. They would almost certainly pass all the standard tests that statisticians use,

for what *that* is worth.

There was a book published by Rand Corporation once which was called ` One Million Random Digits', leading the thoughtful person to enquire whether they were truly random or only appeared to be. How can you tell? The idea that some orders are more random than others is distinctly peculiar, and yet the authors of ` One Million Random Digits' had no hesitation in rejecting some of the sequences on the grounds that they failed tests of randomness . Would the observation that they can't be random because my copy of the book has *exactly the same digits* as yours, allowing me to predict the contents of yours with complete accuracy, be regarded as reasonable? Probably not, but what if an allegedly random set of points in the plane turned out to be a star map of some region of the night sky? All these matters make rather problematic the business of deciding if something is random or not. Nor does the matter of deciding whether something is realio-trulio random allow of testing by applying a fixed number of procedures. And yet randomness appears to be a necessary `condition A' in lots of probabilistic models, from making decisions in a card game to sampling theory applied to psephologists second guessing the electorate.

These points have been made by Rissanen, but should trouble anybody who has been obliged to use probabilistic methods. Working probabilists and statisticians can usually give good value for money, and make sensible judgements in these cases. Hiring one is relatively safe and quite cheap. But if one were to contemplate automating one, in even a limited domain, these issues arise.

The notion of repeatability of an experiment is crucial to classical, causal models of the world, as it is to probabilistic models. There is a problem with both, which is, how do you know that all the other things which have changed in the interval between your so called replications are indeed irrelevant? You never step into the same river twice, and indeed these days most people don't step into rivers at all, much preferring to drive over them, but you never throw the same coin twice either, it got bashed when it hit the ground the first time, also, last time was Tuesday and the planet Venus was in Caries, the sign of the dentist, and how do you know it doesn't matter? If I collect some statistics on the result of some measurements of coin tossing, common sense suggests that if the coin was run over by a train half way through the series and severely mangled, then this is enough to be disinclined to regard the series before and after as referring to the same thing. Conversely, *my* common sense assures me that if another series of measurements on a different coin was made, and the first half were done on Wednesdays in Lent and the last half on Friday the thirteenth, then this is not grounds for discounting half the data as measuring the wrong thing. But there are plenty of people who would earnestly and sincerely assure me that my common sense is in error. Of course, they probably vote Green and believe in Fairies at the bottom of the garden, but the dependence on subjectivity is disconcerting. In assigning some meaning to the term `probability of an event', we have to have a clear notion of the event being, at least in principle, repeatable with (so far as we can determine) the same initial conditions. But this notion is again hopelessly metaphysical. It entails at the very least appeal to a principle asserting the irrelevance of just about everything, since a great deal has changed by the time we come to replicate any experiment. If I throw a coin twice, and use the data to test a probabilistic model for coins, then I am asserting as a necessary part of the argument, that hitting the ground the first time didn't change the model for the coin, and that the fact that the moons of Jupiter are now in a different position is irrelevant. These are propositions most of us are much inclined to accept without serious dispute, but they arise when we are trying to automate the business of applying probabilistic ideas. If we try to apply the conventional notions to the case of a horse race, for instance, we run into conceptual difficulties: the race has never

been run before, and will never be run again. In what sense then can we assign a probability to a horse winning? People do in fact do this, or they say they do and behave as if they do, to some extent at least, so what is going on here? The problem is not restricted to horse races, and indeed applies to every alleged replication. Anyone who would like to build a robot which could read in the results of all the horse races in history, inspect each horse in a race, examine the course carefully, and taking into account whatever data was relevant produce estimates of the probabilities of each horse winning, will see the nature of the difficulties. There is no universally accepted procedure which could do this for the much simpler case of coin tossing.

So the question of what is an allowable measurement gives us a problem, since specifying what is irrelevant looks to be a time consuming job. If the cards are marked, all bets are off on the matter of whether its better to draw two to fill the flush or one for the straight. And how do you list all the other considerations which would make you believe that the argument given above was rendered inapposite? The classical case is much simpler because we can tell whether or not something is being measured properly if we have been trained in the art of recognising a proper measurement, and we can build machines which can implement the measurements with relatively little difficulty. But how do we get a machine to tell if the other guys are cheating in a game of poker? How does it decide if the shuffling process has produced enough randomness? And when does it give up on the model because the data is not in accord with it? Subjectivity may be something engineers are used to, but they haven't had to build it into robots before. But whenever you test a model you have to ensure that the input conditions are satisfied and the measuring process properly conducted.

A fundamental assumption in classical systems, seldom stated, is the stability of the model under variations in the data and the model parameters. If you cast your mind back to the curious discussions of weights joined by zero thickness, weightless threads, hanging over pulleys, which took place in applied mathematics classes in olden times, you may recall having been troubled by the thought that if the threads were of zero width then the pressure underneath them must have been infinite and they would have cut through the pulley like a knife through butter. And they would have to be infinitely strong not to snap, because the force per unit cross sectional area would also have been infinite. What was being glossed over, as you eventually came to realize, was that although the idealised model made no physical sense, it is possible to approximate it with bits of string sufficiently well to allow the results of calculations to be useful. Idealisations are never satisfied in the real world, but sometimes it doesn't much matter.

What it boils down to then is this: you postulate a model and you simultaneously postulate that if the input data and model parameters are varied just a little bit it won't make any great deal of diference to the output of the model. If you are right about this last assumption, you have a classical model, while if not you have a probabilistic model and have to to be less ambitious and settle for knowing only the relative probabilities of the outcomes.

This is complicated by the platonic, metaphysical component of probabilistic modelling: you may be given a set of conditions under which you are told it is safe to use the model, but the conditions are not such as can be confirmed by any operational procedure. For example, you are allowed to make inferences about populations from samples providing your sample is `random', but there is no procedure for ensuring that a given selection scheme *is* random. Or you are assured that using a normal or gaussian distribution is warranted when the process generating the data has a `target' value and is influenced by a very large number of independent factors all of which may add a small disturbing value and satisfy some

conditions. But you are not told, for a particular data set, how to set about investigating these factors and conditions. This situation, in which models are `justified' provided they satisy certain conditions, but there is no operational way of deciding whether the conditions are satisfied or not, is common. The extent to which Statistical methodology is a matter of training people to tell the same story, as in theological college, as opposed to giving the only possible story, as in science, is far from clear, but the assurances one gets from working probabilists are not particularly comforting.

Next Up Previous Contents

Next: Probabilistic Models as Data **Up:** History, and Deep Philosophical **Previous:** Histograms and Probability Density *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Models and Data: Some **Up:** Statistical Ideas **Previous:** Models and Probabilistic Models

Probabilistic Models as Data Compression Schemes

Many of the above problems evaporate if one regards probabilistic models in a different light. Instead of seeing them as competitors of Classical mathematical models, we may regard them as simply data compression schemes. I can, and shall, regard a gaussian *pdf* as simply a compressed way of describing a set of points in $\mathbb{R}^n$. This follows Rissanen's approach, see the references for details. This approach is relatively new, but very appealing to those of us who can almost believe in information theoretic ideas.

As has been pointed out by Penttila, another Finn, the Babylonians kept careful records of eclipses and other astronomical events. They also kept careful records of city fires and floods and famines. The records allowed them to eventually become able to predict eclipses but not to predict fires, floods or famines. Thus they only got good results on the things they didn't actually care about. What is done in practice is to follow the example of the Babylonians: you try hard to use the data to make predictions by any method you can devise. If you can't find any pattern at all, you say to yourself `Sod it, the bloody stuff is random'. This is essentially a statement about your intellectual limitations. If I am much cleverer than you, I might be able to figure out a pattern you cannot. What the moron describes as `bad luck', the smarter guy identifies as incompetence. One man's random variable is another's causal system, as Persi Diaconis can demonstrate with a coin.

This leads to one taking a class of models and asking how much information can each member of the class extract from the given data set? We may distinguish between a class of models which we can write down and where we are able to do the sums to answer the question *how much does this model compress the data*, and a class of models which might be implemented in the neural tissue of a particular human being, concerning which, it must be admitted, little can be said. For each class of models we can imagine a robot implementing that class, and using the data to select a model or subset of models from the class, possibly by maximising the amount of compression of the data. Presumably, human beings, jelly-fish and other elements of creation are also busily trying to compress the data of life's experience into some family of models. It so happens we can say very little about these machines because we didn't build them; the robots we can say a little about. And a robot of a given class will say that some data is random precisely when that robot is unable to effect any significant amount of compression with the family of models at its disposal.

Once one has abandoned attempts to determine whether anything is `realio-trulio' random and observed that randomness is relative to some system trying to extract information, much clarity is gained. The belief that there is some absolute sense of randomness, is one of those curious bits of metaphysics to which certain kinds of impractical, philosophical temperaments are vulnerable, but it makes no particular sense. The definitions of randomness in terms of description length which were initiated by Solomonoff,

Kolmogorov and Chaitin gives no practical decision procedure for declaring something to be random, it only allows you, sometimes, to observe that it isn't. I shall expand upon this later. See the writings of Rissanen for the words of the prophet on these matters.

- Models and Data: Some models are better than others

Next Up Previous Contents

Next: Models and Data: Some **Up:** Statistical Ideas **Previous:** Models and Probabilistic Models *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Maximum Likelihood Models **Up:** Probabilistic Models as Data **Previous:** Probabilistic Models as Data

Models and Data: Some models are better than others

Given a data set and a (generally infinite) set of models, the problem of finding the `best' model in the set to describe the data arises. This is known as *point* or *parameter estimation* in the literature. It is not a very well posed problem, as the term `best' is not defined, and there are several possible meanings it might have. In the next sections we look at the best known contenders for `best' models.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Where do Models come **Up:** Statistical Ideas **Previous:** Models and Data: Some

Maximum Likelihood Models

Given a probabilistic model and a set of data, we can calculate the likelihood that the model `produced' each output value and indeed the likelihood that it produced the whole lot in independent applications of a somewhat mythical generating process. Given two such models, we can find out which is more likely to have produced the actual data. For example, if we threw a coin 10 times and got 8 Heads (H) and 2 Tails (T) in some particular order, we might try the family of models

$$\{q \in [0,1] : p(H) = q\}$$

and decide that the p(H) = 0.5 model is not so likely to have produced the outcome as the model with p(H) = 0.7. The computation is rather simple: The p(H) = 0.5 model has p(T) = 0.5 also, and so the probability of the actual data is $(0.5)^8\,(0.5)^2$. This is, of course, the same as for any other sequence of outcomes, although there are many different ways of getting 8 Heads and two Tails. The probability of having generated the same sequence if p(H) = 0.7 is $(0.7)^8\,(0.3)^2$, and some messing with a calculator gives 0.0009765 for the former and 0.005188 for the latter. So we get more than five times the probability of obtaining 8 Heads and 2 Tails on the second model as on the first.(In any order.)

In fact it is rather likely to cross your mind that the most attractive model is the one with p(H) = 0.8. It is not hard to prove that this particular model gives a bigger value of the probability of getting that output than any other model from that family. I shall call it the *Maximum Likelihood Model*. Note that in this case we have a nice little continuous space of models, each one specified by a number between 0 and 1. I shall refer to this as a one parameter family of models. More generally, it is often the case that we have one model for each point in a manifold or manifold with boundary, and such families of models are referred to as *parametric models*. In some cases it is considered necessary by statisticians to deal with a set of models which cannot be, or at least have not yet been, finitely parametrised and such sets of models are called *non-parametric*.

This invites critical comment based on the observation that life provides only finite amounts of data and the conclusion that one cannot hope to distinguish between too many models in such a case, but I am trying to be good.

For any parametric family of models, that is to say one where we have a manifold each point of which is a model, and for any fixed data set, we can compute a likelihood of each model having produced that data. Thus we have a map from the cartesian product of the manifold and the possible data sets into the Reals. In many cases, for a fixed data set, this function has a unique maximum, that is, there is a unique model for which the likelihood of a fixed data set is a maximum. There is a strong intuitive feeling that induces many people to show a particular fondness for the Maximum Likelihood model, and there are routine ways of computing it for several families of models. For example, given a family of gaussians on

$\mathbb{R}^n$ parametrised by the centres and the covariance matrices, computing the centroid and the covariance for the data gives the maximum likelihood model.

- Where do Models come from?

Next Up Previous Contents

Next: Where do Models come **Up:** Statistical Ideas **Previous:** Models and Data: Some *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Bayesian Methods **Up:** Maximum Likelihood Models **Previous:** Maximum Likelihood Models

Where do Models come from?

It is reasonable to ask in the case of coin tossing, where the model comes from. It may well come out of a hat or the seething subconscious of the statistician analysing the data. If I collect some data and you give me two discrete models, I can find which gives me the larger probability. If they are continuous models, I can, using the idea of the *pdf* as a limit of histograms, calculate the likelihood, the value of the *pdf* at the data value, for each of the data points, and multiply the numbers together to get a total likelihood for the whole data set, on that model. Or as statisticians prefer to think of it, the likelihood of those model parameters for that particular data set. This was discussed, somewhat casually, in the particular case of gaussian models in chapter one. The heuristic rule of picking the larger of the likelihoods was advanced, without anything other than an appeal to intuition to justify it. In the case where the models were all gaussians it is possible to find that model, in the now infinite manifold of all such models, which gives the maximum likelihood. But then we can ask, where does this family of models come from? Maybe gaussians are daft.

There is no algorithm for producing a good family of models; given two models we can produce rules for choosing between them. When the two models are different members of a family with some fixed number of parameters, we can imagine ourselves swanning about the manifold of all such models, looking for the one for which the likelihood of getting our data is maximal. When the models are of very different types, we can still choose the model which gives the maximum likelihood, but this could be a very stupid thing to do. There is always one model which produces exactly the data which was obtained, and no other, with probability one. In the case of the coin which came down Heads 8 times out of 10, there is always the theory that says this was bound to happen. It has certainly been confirmed by the data!

This `predestination' model, easily confused with the data set, has probability 1, if it can be assigned a probability at all, but you are unlikely to feel kindly disposed towards it, if only because it says absolutely nothing about what might happen if you tossed the coin again.

Next Up Previous Contents

Next: Bayesian Methods **Up:** Maximum Likelihood Models **Previous:** Maximum Likelihood Models

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Bayes' Theorem **Up:** Statistical Ideas **Previous:** Where do Models come

Bayesian Methods

Consider a situation such as that of Exercise 1.1, i.e. the first exercise at the end of chapter one. We imagined that we were going to get lots of pictures of half naked women and about ten times as many trees. Your job was to classify a new image. Now you already have odds of 9 to 1 that it's a tree. Shouldn't you use this information in doubtful cases? In this situation, you are treating the models themselves as objects on which to put probabilities. Doing this systematically gives the Bayesian solution, which is an attempt to rationally utilise other sources of information about any one model being the `best'.

More generally, we may have prejudices about the world based on other experiences upon which we are inclined to rely more heavily than a single small amount of data. In the case of the coins, I may have a predisposition to believe that the probability of Heads is much closer to 0.5 than the 0.8 which this single experiment would suggest. This may reflect my having thrown very similar looking coins more than ten times in the past and having got much less than 80% of them coming Heads. I do not approach this experiment in a total vacuum. These considerations lead to Bayesian ideas about how to choose the `best' model.

- Bayes' Theorem
- Bayesian Statistics
- Subjective Bayesians

Next Up Previous Contents

Next: Bayes' Theorem **Up:** Statistical Ideas **Previous:** Where do Models come *Mike Alder*
9/19/1997

Bayes' Theorem

Suppose we are given the *codomain* of a discrete random variable, that is to say the set of *k* possible outcomes. Each such outcome will be referred to as an *atomic event*. I shall use the labels 1 to *k* to specify each of the outcomes.

If we imagine that the *rv* is used to generate data sequentially, as with succeeding throws of a coin, then we will obtain for a sequence of *n* such repetitions a space of k^n distinct outcomes. The fact that these are repetitions of the same *rv*, means that each of the *n* repetitions is a separate event in no way contingent upon the results of earlier or later repetitions. The same histogram should be applied to each of them separately when it comes to computing probabilities. Then this defines us a particular kind of new *rv* which is from the cartesian product of *n* copies of the domain of the basic *rv*, and goes to the cartesian product of *n* copies of the codomain, by the obvious product map. It follows that the probability of an atomic event in the product *rv* is the product of the probabilities of the components.

There may be a different *rv* on the same domain and into the same codomain(k^n) in the case where what happens at the j^{th} stage in the sequence depends upon what happens at earlier (or later) stages in the sequence. In such a case the probability of an event in the codomain will not usually be expressible as such a product. In order to deal with such cases, and for other situations also, the idea of *conditional probability* is useful.

For any *rv*, an *event* will be used to mean a non-empty element of the power set of the space of atomic events, i.e. a non-empty subset of the codomain of the *rv*. For example, if the *rv* is the obvious model for the throwing of a six sided cubic die, the atomic events can be labelled by the integers 1 to 6, and the set of such points $\{1,3,5\}$ might be described as the event where the outcome is an odd number. Then

events inherit probabilities from the atomic events comprising them: we simply take the measure of the inverse image of the set by the map which is the *rv*. Alternatively, we take the probability of each of the atomic events in the subset, and add up the numbers.

Suppose *A* and *B* are two events for a discrete random variable. Then $p(A)$ makes sense, and $p(B)$ makes sense, and so does $p(A \cup B)$ and $p(A \cap B)$. It is clear that the formula

$$p(A \cup B) = p(A) + p(B) - p(A \cap B)$$

simply follows from the fact that p is a measure, i.e. behaves like an area or volume. In addition, I can make up the usual definitions:

Definition: *A* and *B* are *independent* iff $p(A \cap B) = p(A)p(B)$

It might not be altogether trivial to decide if two events which are observed are produced by a random variable for which they are independent. Usually one says to oneself `*I* can't think of any reason why these events might be causally related, so I shall assume they aren't.' As for example when A is the event that a coin falls Heads up and B is the event that it is Tuesday. It seems a hit or miss process, and sometimes it misses.

Definition: If *A* and *B* are any two events for a discrete *rv* the *conditional probability of A given B*, $p(A|B)$, is defined when $P(B) \neq 0$ by

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

There is a clear enough intuitive meaning to this: Given that event *B* has occurred, we really have to look at the measure of those points in the domain of the *rv* which can give rise to an outcome in the event *A* from among those which can give rise to the event *B*. In short we have what a category theorist would call a *subrandom variable*, were category theorists to be let loose in probability theory. A straightforward example would be when *B* is the event that a die is thrown and results in a value less than four, and *A* is the event that the die shows an odd number. Then $p(A|B)$ is the probability of getting one or three divided by the probability of getting one or two or three, in other words it is $\frac{2}{3}$. We can also calculate $p(B|A)$, which is the probability of getting one or three divided by the probability of getting one or three or five, which is also $\frac{2}{3}$. In general the two probabilities, $p(A|B)$ and $p(B|A)$, will be different. In fact we obviously have the simple result:

Bayes' Theorem $p(B|A) = \frac{p(A|B)p(B)}{p(A)}$

This follows immediately from the rearrangements:

$$p(A|B)p(B) = p(A \cap B) = p(B|A)p(A)$$

It would be a mistake to imagine that Bayes' theorem is profound, it is simply linguistic. For some reason, Bayes got his name on a somewhat distinctive approach to deciding, *inter alia* which model is responsible for a given datum. For now let us suppose that this extends to the case of continua, and we have the situation of *Fig.1.10* of chapter one. Here, two models, g_m and g_f perhaps, are candidates for having been responsible for the point *x*. We can calculate without difficulty the numbers $g_m(x)$ and $g_f(x)$. We decide to regard these two numbers as something like the conditional probability of having got *x* given model *m* and the conditional probability of having got *x* given model *f*, respectively. We may write

this, with extreme sloppiness, as $p(x|m)$ and $p(x|f)$ respectively. These would be interpreted by the sanguine as measuring the probability that the datum x will be observed given that model m (respectively, f) is true. Now what we want is $p(m|x)$ and $p(f|x)$ respectively, the probabilities that the models are true given the observation x. By applying Bayes Theorem in a spirit of untrammelled optimism, we deduce that

$$p(m|x) = \frac{p(x|m)p(m)}{p(x)}$$

with a similar expression for $p(f|x)$.

Suppose we are in the situation of the first problem at the end of chapter one; you will doubtless recall the half naked ladies and possibly the trees from which they were to be distinguished. Now it could be argued that in the absence of any data from an actual image, we would rate the probability of the image being of a tree as 0.9 and of it being a naked lady as 0.1, on the grounds that these are the ratios of the numbers of images. Bayesians refer to these as the *prior probabilities* of the events. So in the above formulae we could put $p(m)$ and $p(f)$ in as numbers if we had some similar sort of information about the likelihoods of the two models. This leaves only $p(x)$ as a number which it is a little hard to assign. Happily, it occurs in both expressions, and if we look at the *likelihood ratio*, it cancels out. So we have:

$$\frac{p(m|x)}{p(f|x)} = \frac{p(x|m)p(m)}{p(x|f)p(f)}$$

and the right hand side, known as the *likelihood ratio* is computable. Well, sort of.

If you are prepared to buy this, you have a justification for the rule of thumb of always choosing the bigger value, at least in the case where $p(m)$ and $p(f)$ are equal. In this case, $p(m|x)$ is proportional to $p(x|m)$ and $p(f|x)$ is proportional to $p(x|f)$ and with the same constant of proportionality, so choosing whichever model gives the bigger answer is the Bayes Optimal Solution. More generally than the crude rule of thumb, if it is ten times as likely to be m as f which is responsible for a datum in a state of ignorance of what the actual location of the datum is, then we can use this to obtain a bias of ten to one in favour of m by demanding that the ratio of the likelihoods be greater than ten to one before we opt for f as the more likely solution.

The blood runs cold at the thought of what must be done to make the above argument rigorous; the step from probabilities, as used in the formulae, to likelihoods (since things like $p(x|m)$ interpreted as $g_m(x)$ are distinctly shonky) requires some thought, but reasonable arguments about *pdf*s being really just the limits of histograms can carry you through this. What is more disturbing is the question of what the random variable is, that has events which are made up out of points in $\mathbb{R}^n$ together with gaussian (or other) models for the distribution of those points. Arguments can be produced. A *Bayesian* is one for whom these (largely philosophical) arguments are convincing. The decision arrived at by this method is known as the *Bayes Optimal decision*. Sticking the word `optimal' in usually deceives the gullible into the conviction that no critical thought is necessary; it is the best possible decision according to Bayes, who was a clergyman and possibly divinely inspired, so to quarrel with the Bayes optimal decision would be chancing divine retribution as well as the certainty of being sneered at by an influential school

of Statisticians. Rather like the calculation of ` 99% confidence intervals' which suggest to the credulous that you can be 99% confident that the results will always be inside the intervals. This is true only if you are 100% confident in the model you are using. And if you are, you shouldn't be.

Using power words as incantations is an infallible sign of ignorance, superstition and credulity in the part of the user. It should be left to politicians and other con-men.

Next Up Previous Contents

Next: Bayesian Statistics **Up:** Bayesian Methods **Previous:** Bayesian Methods *Mike Alder*
9/19/1997

Bayesian Statistics

In the decision case of telling the guys from the gals, we assumed that the models *m* and *f* were given and non-negotiable and that the problem was to choose between them as the most probable explanation for the occurrence of the datum *x*. In many problems, we have the data and the issue is to find the best or most probable *pdf* to account for the whole lot. *Point or parameter estimation* is the business of finding the `best' model for a given data set. The question, of course, is, what does `best' mean? We have discussed the idea of the Maximum Likelihood estimator, but there are cases when it is obvious even to the most partisan that it is not a particularly good choice. In the case of the coin which was thrown ten times and came down Heads eight times, the Maximum Likelihood model leads you to believe it will come down 800 times, give or take a few, if it is tossed a thousand times. This might surprise someone who looked hard at the coin and concluded that it looked like any other coin. Such a person might be prejudiced in favour of something closer to 0.5 and unwilling to go all the way to 0.8 on the strength of just ten throws.

Let *m* be the model that asserts $p(H) = m$; then there is a model for each $m \in [0,1]$. Let *x* be the

observation consisting of a particular sequence of 8 Heads and 2 Tails. Then $p(x|m) = m^8(1\text{-}m)^2$ for any such *m*. If we maximise $p(x|m)$ for all values of *m*, we get the maximum likelihood model, which by a bit of easy calculus occurs when

$$8m^7(1\text{-}m)^2 - m^8(2)(1\text{-}m) = 0$$

i.e. for $m = 0.8$, as claimed in *3.1.5.*

Now let us suppose that you have a prejudice about the different values for *m*. What I shall do is to assume that there is a probability distribution over the different values which *m* can have. If I try the distribution $6m(1\text{-}m)$ then I am showing a preference for the value of *m* at 0.5 where this *pdf* has a maximum, but it is not a very powerful commitment.

Now looking at the Bayes formula

$$p(m|x) = \frac{p(x|m)p(m)}{p(x)}$$

we obtain immediately that

$$p(m|x) = \frac{m^8(1-m)^2 6m(1-m)}{p(x)} = \frac{6m^9(1-m)^3}{p(x)}$$

Since $p(x)$ is some prior probability of x which, I shall suppose, does not depend upon the model, m, I can use calculus to obtain the maximum of this expression. A little elementary algebra gives a value of $m = \frac{3}{4}$. This is between the maximum likelihood estimate and the prior prejudice of 0.5. Since my commitment to 0.5 was not very strong, as indicated by the *pdf* I chose, I have gone quite a long way toward the ML model. Had I used $30m^2(1\text{-}m)^2$ as my measure of prejudice, I should have been closer to 0.5.

The above argument assumes that $p(x)$ does not depend on m. But what *is* $p(x)$? If x has just been observed, then it ought to be 1; so it doesn't mean that. The most sensible answer is that it is the so called *marginal* probability, which is obtained by adding up all the $p(x|m)$s for all the different m, weighted by the probability of m. Since these are all the values between 0 and 1, we have

$$p(x) = \int_{m=0}^{m=1} p(x|m)p(m)dm$$

which means that dividing by $p(x)$ simply amounts to normalising the numerator in the definition of $p(m|x)$ above, thus ensuring that it is a *bona fide pdf*. So we have not just a most probable value for the estimated value of the parameter m given the observation data x, we have a *pdf* for m. This *pdf* known, not unreasonably, as the *posterior pdf*, and is written, as above, as $p(m|x)$, and may be compared with $p(m)$ as the *prior pdf*. It is easy to compute it explicitly in the case of the coin tossing experiment, and the reader is urged to do this as it is soothing, easy and helps burn the idea into the brain in a relatively painless manner. The maximum of the posterior *pdf* is at some places in the literature referred to as the MAP probability, because acronyms are much less trouble to memorise than learning the necessary latin, and *Maximum A Posteriori* takes longer to say than `MAP', even though it can be difficult for the untrained ear to hear the capital letters.

In the general case where data is coming in sequentially, we can start off with some prior probability distribution, and for each new datum use this method to predict what the datum ought to be, and when we find out what it actually was, we can update the prior. This is clearly a kind of learning, and it is not beyond the range of belief that something similar may occur in brains. We talk, at each stage of new data acquisition, of the *a priori* and *a posteriori pdf*s for the data, so we obtain an updated estimate of the `best' *pdf* all the time. This will not in general be the same as the Maximum Likelihood estimate, as in the example.

Next Up Previous Contents

Next: Subjective Bayesians **Up:** Bayesian Methods **Previous:** Bayes' Theorem *Mike Alder 9/19/1997*

Subjective Bayesians

There are people who feel in their bones that it makes sense to assign a prior distribution to such things as the different models for the coin tossing in the last section, so as to represent their own internal states of belief. Such a person might say `Well, I think that $p(H) = 0.5$ is reasonable as a result of looking at the coin and its symmetry. I suppose I could be persuaded that $p(H) = 0.4$ or $p(H) = 0.6$ are reasonably likely, while I am much more sceptical about values outside that region. The preceding sentences express my prejudices in English, which is a very scruffy and imprecise language, what I *really* mean is that my degree of belief in the model $p(H) = m$ is precisely $6m(1-m)$ for m between 0 and 1. '

This kind of statement may strike one as rather odd. It suggests, at the very least, a good deal of confidence in one's powers of introspection. Most people can barely make up their minds what to have for dinner tonight, while a subjective Bayesian appears to have his preferences expressible by analytic functions. The idea that beliefs can be expressed with such precision and certainty, to as many decimal places as one wishes to use, has been criticised severely by, for instance, Peter Walley.

We may doubt that an individual's beliefs can be known to him with such precision, but if we regard any animal as a learning machine which at any time has amassed a certain amount of knowledge of how the world works, then any situation in which it finds itself might be said to be a generator of expectations of what will happen next. These expectations are, one presumes, the result of the animal's brain having stored counts of what happened next, in roughly similar circumstances, in the past. How the brain stores and uses this information is of course still largely conjectural, but it would appear to be what brains are for. And it is not beyond belief that the different possible outcomes of a situation may be assigned different degrees of confidence in some representation accomplished by neurons, so there may indeed be some discrete approximation to a Bayesian prior *pdf* sitting in some distributed fashion in the brain of the animal. And if the animal happens to be a subjective Bayesian, he may feel competent to tell us what this *pdf* feels like, even unto using the language of analytic functions. This last is the hardest part to believe, but dottier ideas have come good in Quantum Mechanics, as my old Mother used to say.

When the animal is fresh out of the egg, it may have rather bland preferences; these may be expressed by uniform priors, for example the belief that all values of m are equally likely. Or we may suppose that far from having a *tabula rasa* for a brain, kindly old evolution has ensured that its brain has in fact many pre-wired expectations. If an animal is born with wildly wrong expectations, then kindly old evolution kills it stone dead before it can pass on its loony preconceptions to junior.

It is not absolutely impossible then that at some future time, when the functioning of brains is much better understood than it is at present, by some careful counting of molecules of transmitter substances at synapses using methods currently undreamt of, it might be possible to actually measure somebody's prior *pdf* for some event. Whether this might be expected to give results in good agreement with the opinion of the owner of the brain on the matter is open to debate. Subjective Bayesians propose to carry on as if the experiments had been carried out and given the answers they want to hear; this seems foolhardy, but one

can admire their courage.

Objective Bayesians are in Schism with Subjective Bayesians; just as the Fascists were in schism with the Communists. And it is pretty hard for outsiders to tell the difference in both cases, although the insiders feel strongly about them. Jaynes has written extensively about sensible priors used to describe ones ignorance or indifference in a way which should be carefully studied by anyone with a leaning toward Fuzzy Sets. An objective Bayesian believes that one can work out not what prior is in his head, but what prior *ought* to be in his head, given the data he has. Thus he has a commitment to the view that there really is a best thing to do given one's data and the need for decision. If the subjective Bayesian believes he is modelling his own belief structures, the objective Bayesian is modelling what he thinks he ought to believe. Presumably he believes he ought to believe in objective Bayesianism; the possibility of an objective Bayesian concluding that there is insufficient evidence for objective Bayesianism but deciding to believe in it anyway, ought not to be ruled out. The idea that there is a best prior, a right and reasonable prior, but that it might be very difficult to find out what it *is*, leads to *sensitivity analysis* which aims to determine if a particular choice of prior is likely to lead you into terrible trouble, or if it doesn't much matter which prior you choose from a class.

It is much easier to philosophise in a woolly manner, and get nowhere, than to come to clear conclusions on matters such as these. This makes the subject fascinating for some and maddening for others.

I.J. Good has written on the philosopical issues involved, if you like that sort of thing.

Next Up Previous Contents

Next: Minimum Description Length Models **Up:** Bayesian Methods **Previous:** Bayesian Statistics *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Codes: Information theoretic preliminaries **Up:** Statistical Ideas **Previous:** Subjective Bayesians

Minimum Description Length Models

So far we have considered the issue of selecting, from some family of models, that model which is `best' by considering which model assigns the data the higher probability, possibly bearing in mind prior prejudices about reasonable results. Another issue in comparing two models is not just the likelihood of the data having arisen from the model, but the model complexity.

Simple models which give reasonably close results are more attractive that very complicated models which give exactly the right answer. In fact we may decline to believe a sufficiently right model, and we always feel uncomfortable with very complicated ones. So model complexity is a consideration, and the question of how we measure it arises naturally. This is where Rissanen comes in.

Rissanen has argued roughly as follows: a model of a data set allows you to compress the data for purposes of transmission or storage. If, for example you had a data set consisting of a sequence of 100 digits which went 314159.... and agreed with π until the end of the sequence, sending the information coded as `first hundred digits of π, forget the decimal point' would compress the sequence rather considerably. As long as the guy at the other end knew what π meant. If you don't feel safe about this, you may feel obliged to tell him, by, for example, also sending the algorithm for generating π. This extra, I shall call the `overhead'. Now one model is better than another in Rissanen's sense if it does more compression, taking into account the overhead. If you reason that the more you compress the data, the more information you are extracting from it, this sounds very plausible. This principle allows us to reject the maximum likelihood `raw data' model in favour of a gaussian model when the number of data points exceeds some quite small number. It also allows us to make sensible decisions on how many gaussians one ought to use in modelling a point set as a mixture of gaussians, a matter of practical importance when we get to syntactic pattern recognition. The ideas are related to Chaitin's attempts to characterise randomness. See Chaitin (1975) and Chaitin (1987) for an antidote to some of the cynicism of the present and an earlier section.

In this section I sketch the recently developed and indeed still developing theory of Rissanen on *Stochastic Complexity*, as described in his book Rissanen(1989) and papers (see the bibliography). It offers a philosophically quite different approach to finding the `best' model to account for the data, and an approach which avoids some of the objections I have pointed out to the classical theory. In particular it appears to solve elegantly the issue of how many gaussians one ought to have in a gaussian mixture model for a particular data set. (The answer may be `none', if the data set is small enough!) This will concern us in the next chapter, where we meet such beasts.

The general question as to order of a model occurs in other settings also. At the simple level of fitting a polynomial to a set of data points, the question as to what degree of polynomial is appropriate arises. At one extreme one can have the degree so high that every point is fitted, at the other we can assume it is

constant. The same problem arises in the case of Neural Nets, as we shall see in a later chapter.

The gaussian mixture modelling problem, how many gaussians should we use, makes the problem quite stark. There will be an increase in the log-likelihood of the data with respect to the model whenever we increase the number of gaussians, until the degenerate case of non-invertible covariance matrices makes it infinite. But there is a penalty to be paid in terms of the number of parameters needed to specify the increasing number of gaussians. It is sensible to try to penalise the model in some way by offsetting the increase in log-likelihood with a corresponding increase in the number of parameters, and seeking to minimise the combination. But then the question arises, what should be the rate of exchange between log-likelihood and the parameter count? Rissanen gives us a natural and plausible rate of exchange. Because the ideas have not yet percolated through to the Pattern Recognition community in its entirety, I shall explain them in elementary terms.

I would like to be able to assume a moderate familiarity with Shannon's *Information Theory*, which is by now an essential part of the mental furniture of anybody with pretensions to an education, but it seems unsafe to do so. Typically, an engineers mathematical education seems to consist increasingly of having memorised some recipes and then forgotten them, and computer scientists, so called, have left out most mathematics except a little finitary material such as logic and search algorithms. So I shall explain the basics as I proceed. If I insult your intelligence and knowledge then I put my trust in your tolerance.

- Codes: Information theoretic preliminaries
- Compression for coin models
- Compression for *pdf*s
- Summary of Rissanen Complexity

Next Up Previous Contents

Next: Codes: Information theoretic preliminaries **Up:** Statistical Ideas **Previous:** Subjective Bayesians

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Compression for coin models **Up:** Minimum Description Length Models **Previous:** Minimum Description Length Models

Codes: Information theoretic preliminaries

Suppose I have a piece of English text to transmit down a line or store on disk. I shall suppose that the conventional ASCII character set and its usual representation by seven bits is well known to you and everybody who might take an interest in computers and text. Let us assume that the text consists of the sentence

```
the cat sat on the mat
```

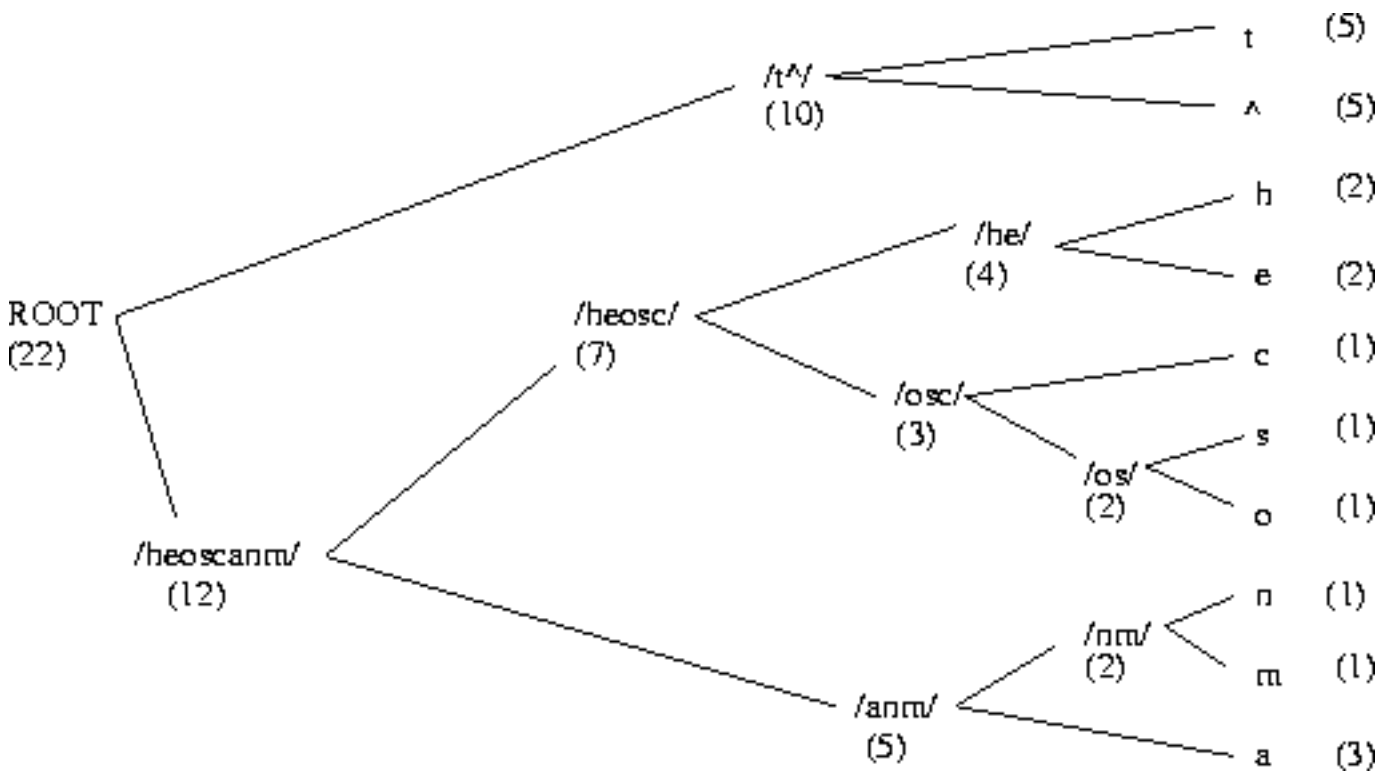
which has just 22 characters. It therefore takes up 154 bits with the usual ascii coding. If we use 8 bits to a character, with say a parity check bit, then it goes up to 176 bits.

Let us examine the best compression we can think of, and ignore the noise and corruption which the parity check bit is designed to detect.

Most letters do not occur at all, and of those that do, the frequency varies. The frequencies of occurrence and the corresponding probabilities (divide by 22) rounded are:

```
t 5  0.22727
^ 5  0.22727
a 3  0.13636
h 2  0.09091
e 2  0.09091
c 1  0.04545
s 1  0.04545
o 1  0.04545
n 1  0.04545
m 1  0.04545
```

Figure 3.3: A Huffman tree to code `the cat sat on the mat'.



I have put the characters in decreasing order of frequency, with a space denoted by ^ . Now I shall recode the characters by giving a new dictionary chosen so that the symbols which occur most often get the shortest binary coding. There are several well known ways of doing this, Shannon-Fano and Huffman being the best known.

In Huffman coding (Huffman, 1952) we imagine that we have the letters and probabilities at the leaves of a tree and proceed to join them together pairwise until we have a single node, the root of the tree. The simplest rule for joining nodes, and the one I shall use in this example, is to pick the two with the lowest summed probability, and to choose arbitrarily when there are several such.

In the above case we join together m and n as these have smallest probabilities and call the resulting node/mn/. Next we join o and s to get /os/. Each of /mn/ and /os/ have sum probability 0.09091 (after allowing for rounding). Now we join /os/ to c to get /osc/ with probability 0.13636. We could have chosen to join /mn/ to c, it doesn't make any difference to the end compression. We continue in this way until we have the whole tree, as indicated in *Fig.3.3.*

Now we obtain a code for each symbol by starting off from the ROOT and proceeding to the symbol. Every time we turn left at a fork we write `0' and every time we turn right we write `1'. Thus the symbol t gets coded as 00, while a space is 01. `a' becomes 111, and the rest are as shown:

```
t 5  0.22727   00
^ 5  0.22727   01
a 3  0.13636   111
h 2  0.09091   1000
e 2  0.09091   1001
```

```
c 1  0.04545   1010
s 1  0.04545   10110
o 1  0.04545   10111
n 1  0.04545   1100
m 1  0.04545   1101
```

Now suppose we code using this system. We get:

```
00 1000 1001 01 1010 111 00 01 10110 111 00 01

t   h    e    ^   c   a  t  ^    s    a   t ^


10111 1100 01 00  1000 1001 01  1101 111 00
  o     n  ^   t    h   e   ^     m    a  t
```

Several points merit attention. First, the total number of bits has dropped from 154 to 67, a considerable saving. Second, the code is a *prefix code* which is to say it can be decoded left to right without putting spaces in, no codeword is a left segment of another codeword. Third, the saving is illusory since I also have to send the alphabet of ten symbols (in ascii, 7 bits per symbol) followed by the code word for it, a total of 70 + 37 bits, which gives 174 bits. On the other hand, we didn't consider the overhead involved in the ascii code being sent!

The need to send the coding scheme I shall refer to as the *overhead* associated with the scheme. Of course, the coding scheme need be sent only once, and it is unreasonable to expect much compression in such a short string anyway. The actual example is fairly silly of course, but using the same idea, it is easy to get respectable levels of string compression for natural language text.

A little thought suggests some improvements. The `model' we are tacitly applying here is for English text being a random sequence of letters each with a predetermined frequency. English is pretty bad, but not as bad as that. If we were to store the frequency of consecutive pairs of symbols, the `bigrams', to use Shannon's terminology, we would get a more accurate model of English. For large enough samples of English text, an accurate bigram count would surely do a better job than the singleton count used in the example. After all, most bigrams don't occur at all, and some redundancies in English (such as u after q except in QANTAS) can be exploited. And why not move on to trigrams and tetragrams and so on? Given large enough samples of text from which to extract the *n*grams, we could extract, you might think, every scrap of redundancy in English. It is true, as one's intuitions suggest, that there are improvements of coding which, by treating blocks of text, get closer to the ideal coding implied by the model, and also that one cannot do better without a better model.

There are several ways of doing better than Huffman coding, but the first point to which I wish to draw your attention is that there are two components to compression of symbols: the first is a *probabilistic model* which gives frequencies of occurences which may be expected of symbols or blocks of symbols. The second is the exploitation of this model by assigning strings of bits to the symbols or blocks so that the high frequency symbols or blocks get the shortest strings.

The second point to observe is that if x is a symbol in an alphabet $\mathcal{X}$ of cardinality M, and if $p(x)$

denotes the probability of getting the symbol *x* in the stream we wish to transmit (or store) according to a model, then Huffman coding gets us reasonably close to being able to code *x* with a string of bits of length *l*(*x*), where $l(x) \approx \log_2(\frac{1}{p(x)})$. It is easy to see that if we deal with *n*grams, the increase in the alphabet size allows us to get closer to this ideal coding length. And given any coding in bits of length *l*(*x*), then the code gives a `probability' to *x* of $(\frac{1}{2})^{l(x)}$. This is shown for the silly example, above, to indicate how one gets reasonably close despite the smallness of the data set.

```
t 5  0.22727   00     0.25
^ 5  0.22727   01     0.25
a 3  0.13636   111    0.125
h 2  0.09091   1000   0.0625
e 2  0.09091   1001   0.0625
c 1  0.04545   1010   0.0625
s 1  0.04545   10110  0.03125
o 1  0.04545   10111  0.03125
n 1  0.04545   1100   0.0625
m 1  0.04545   1101   0.0625
```

If we code $\mathcal{X}$ in the obvious, `blind' way, we need $\log_2(|\mathcal{X}|)$ bits per symbol and if the message is of length *L*, it takes up $L \log_2(|\mathcal{X}|)$ bits to send it, neglecting overhead. If we code it using the model, we have, assuming we can get *l*(*x*) close to ideal,

$$L \sum_{x \in \mathcal{X}} p(x) \log_2(\frac{1}{p(x)})$$

which is usually less. In fact it cannot be more, and the two are equal in the `maximum entropy' case where $p(x) = \frac{1}{|\mathcal{X}|}$. Proving this is a simple bit of calculus. Note that we take the value of $p(x) \log_2(\frac{1}{p(x)})$ to be 0 when *p*(*x*) is zero- which makes sense if you draw the graph of $p(x) \log_2(\frac{1}{p(x)})$ for x between 0 and 1.

The third point to observe is that if you poured water at 1 litre per second into the root of the tree in *Fig.3.6* and the water splits up at the first node so that half goes each way, and similarly for every

subsequent place where the pipe bifurcates, then the water coming out at the leaf labelled t emerges at $\frac{1}{4}$ litres per second, and if *x* is a leaf, then the amount of water coming out at *x* is $\frac{1}{2^{l(x)}}$. Now if you add up the amount of water coming out of all the leaves, equivalent to adding up all the numbers in the last column of the above table, you get the amount of water going in, which is 1. It is clear that you couldn't have more water coming out than going in, but the above counting neglects the possibility that we might have some code words left over with no symbols to be assigned to them. We therefore deduce the rather trivial result:

$$\sum_{x \in \mathcal{X}} \frac{1}{2^{l(x)}} \leq 1$$

It is easy to see that there is a generalisation to the case where we use more than two symbols in place of our binary code. We leave it to the ingenuity of the reader to write it down. This result is the well known *Kraft Inequality*. It is plain that any code for $\mathcal{X}$ must satisfy the Kraft inequality, and this sometimes comes in useful.

A fourth point to note is that if the model, which asserts something about the probability of getting the symbol *x*, is wrong, then it is easy to see that the coding based upon the model will be less effective at compressing the data stream than if the model were right. This follows because we can compare the length of the coded data stream using the actual frequencies, say *q*(*x*), when we get

$$L \sum_{x \in \mathcal{X}} q(x) \log_2 \left(\frac{1}{p(x)} \right)$$

for the length of the bitstream coded on model *p* and

$$L \sum_{x \in \mathcal{X}} q(x) \log_2 \left(\frac{1}{q(x)} \right)$$

for the length of the bitstream coded on model *q*, and the second is less than the first by an easy little exercise in constrained optimisation. It is obviously a good idea to get as good a model as possible for what actually happens, if you want the maximum compression.

A fifth point to make is the rather obvious one that if we can predict the rest of the string past some point with absolute certainty, then the rest of the string is not informative to anyone who knows our prediction system. All we need to do is to send our prediction system and that suffices to allow the receiver to reconstruct the rest of the sequence. This, of course, is the idea behind sending the algorithm for π instead of the first ten thousand digits. Our probabilistic model has some probability 1, and the length of the compressed string falls to zero. We have to send only the overhead.

Summarising, compression arises from redundancy in the string which allows us to model the string and

the different frequencies of the components of the string. The more we know about the string the shorter the resulting coded version. The belief that there is some intrinsic amount of redundancy in the string is simple minded, what we have is the possibility of better and better models. The better your model, the more compression. We can extract redundancy from the data, relative to the model, by a coding scheme. The difference

$$L\left(\log_2(|\mathcal{X}|) - \sum_{x\in\mathcal{X}} p(x)\log_2(\frac{1}{p(x)})\right)$$

measures the amount of information extracted from a stream of symbols from the alphabet $\mathcal{X}$ of length L by the model which gives the probabilities over the alphabet. In the case where the model fits the data perfectly with probabilities being 1 or 0, this is all of the information there is.

In practice, in order to use any such scheme, we have to transmit the model itself, which overhead may make it simpler to transmit the data instead. For large amounts of data, compression would seem to be a good investment.

This gives us an unambiguous sense of when one model is better than another: we can measure the amounts of compression we get on the data which the model is supposed to be modelling. The one which gives higher compression has been extracting more information (that is the only sensible way to measure information) from the data. In making this comparison, we may have to consider the overhead of the models. If the models have the same number of parameters, then the overheads are the same and may be ignored, but if the model is complex, the overhead may be unacceptable- and this may be determined unambiguously by measuring the total compression, taking into account the overhead.

You will be, I hope, familiar with the practicalities of *Information Theory*, or *The Mathematical Theory of Communication*, as Shannon prefers to call it. In order to think about what is going on, as opposed to simply knowing how to do the sums, it is useful to think of the basics as follows.

First one argues that information is passed in *messages* which consist of temporal sequences of *symbols*. We may as well take it that these symbols come from some finite alphabet, because if they came from an infinite alphabet, either we should have to wait an infinite length of time to get just one of them or we should have some practical limit on our ability to distinguish when two of them are different. So we model the whole business of communication as an exchange of symbol strings from some finite alphabet. Now the sender and receiver of these strings, not necessarily different people, will have, at any point in the string, some kind of expectation about the possibilities of what symbol can come next, which we propose to model as a probability distribution over the alphabet. This probability distribution will, of course, change as we go down the string of symbols.

Now at any point, for any symbol in the alphabet, I shall take the probability of the symbol and take its reciprocal and call it the *improbability*. Then I take the logarithm of this improbability and call it the *surprise* of the symbol turning up at this location. This seems a reasonable name, since if the probability is 1, so it is certain, then the improbability is also 1 and the logarithm is zero, so I shall get zero surprise. If the probability is 0 so that I firmly believe it can't possibly happen, then I get an infinite amount of surprise, which serves me right. Not so much a surprise as a paralysing shock. And if the probability is 0.5, then the improbability is 2, and if I take logarithms to the base 2, I get one unit of surprise. Such a

unit is called a *bit*. Since probabilities for independent events multiply, so do improbabilities, and surprises simply sum. So if I get one bit of surprise from one throw of a coin (as I will if it is a fair coin, or if I believe it is) and then I throw it again and get another bit of surprise, I get altogether two bits of surprise.

Note that this is the surprise produced by the data *relative to the model*. It doesn't make sense to talk of the amount of surprise in the data *per se*. If you and I have different models, we might experience quite different amounts of surprise at different parts of the string of symbols.

The average amount of surprise I expect is easily calculated: the symbol x will occur with frequency $p(x)$ (which will be conditional on all sorts of things) and each time it occurs I get $\log_2(\frac{1}{p(x)})$ bits of surprise, so the total is the sum of this over all symbols in the alphabet, weighted by the relative frequency of occurrence. This gives the well known formula for the *entropy* of the probability distribution:

$$\sum_{x \in \mathcal{X}} p(x) \log_2(\frac{1}{p(x)})$$

This is clearly a property of the model and has nothing at all to do with the data. It is, of course, known as the *entropy* of the model. I can also compute, by summing the amount of surprise as I go along, the total amount of surprise I get from any given message, and the average surprise per symbol requires that I divide this by the number of symbols in the message. If the model is a good model of the message, then these two numbers ought to be fairly close. Given two models, I can measure the exent to which they expect, on average, the same symbols with similar probabilities, and thus get a measure of how different the models are, the *Kullback-Leibler* measure. What I call the `surprise' is more commonly called the *information* associated with the symbol. I guess it sounds classier.

Most of this may be found, in improved and extended form, in the book by Shannon and Weaver, which should be on everybody's bookcase. What isn't may be found in the book *Elements of Information Theory* by Cover and Thomas of the bibliography.

In conclusion, a thought for the day.

Brains may be regarded as devices for figuring out what is likely to happen next so as to enable the owner of the brain to improve his chances of survival. A brain has to compress the experience of the owner's life so far, and possibly also some of the experience of his ancestors. The amount of data is huge, and compressing it is essential. Happily, the better at extracting information from that experience, the more compression. So a device for figuring out what is likely to happen next will have the best chance of surviving in a hostile world when it is best at extracting information and hence of compressing its experience.

Next Up Previous Contents

Next: Compression for coin models **Up:** Minimum Description Length Models **Previous:** Minimum Description Length Models *Mike Alder*

9/19/1997

Compression for coin models

Suppose we conduct the experiment with tossing a coin ten times and get the sequence of 8 Heads and 2 Tails, as described earlier. Again contemplate two models, the p(H) = 0.8 model and the p(H) = 0.5 model. I want to compute the compression I can obtain by using these models to describe the actual data.

Using model m=0.5, because the arithmetic is easier, on the first throw, the probability of Heads (according to the model, models are things that have probabilities, not coins) is 0.5. The improbability is the reciprocal of this which is 2, and the information I get when a Head actually turns up is $\log_2(2) = 1$. This happens for the first 8 throws of the coin, and I get 1 bit of information or surprise each time. On the last two throws, I get a Tail which I also expect with probability 0.5, so I get another 1 bit of information for each of these. This adds up to a total of ten bits, and that is the most uninformative model I could have for the process.

Using model m = 0.8, I get a total amount of surprise of

$$8\log_2(\frac{1}{0.8}) + 2\log_2(\frac{1}{0.2}) = 8(0.3219) + 2(2.3219) = 7.219$$

which is less. So the amount of compression of this string of digits on this second model could reduce it from 10 bits to just over 7 bits. Huffman coding won't help, there are too few symbols, but *Arithmetic coding* can help. I describe this now.

In writing a number between 0 and 1 in binary, I imagine representing my number as a point on the line. I divide the line into two equal bits, with $\frac{1}{2}$ in the middle. If the number is in the left half of this partition, I write 0, if in the right half I write 1. Now I take the half containing the point and blow it up so that it looks the same as the original interval, and again I subdivide it in the middle. If my point is in the left half I write 0, if in the right half I write 1. So far I have two digits, .00 if the point is in the left quarter, .01 if it lies between $\frac{1}{4}$ and $\frac{1}{2}$, and so on. I continue in this way until I have specified the position of the point to a sufficient precision, unless I am a pure mathematician in which case I continue indefinitely. This gives pure mathematicians something to do.

Now there was no necessity to chop up the interval into two subintervals; had I used ten and labelled them from 0 to 9, I should have obtained the decimal expansion of the number by the same process. Nor is there any necessity to chop the interval up into *equal* intervals. I shall now chop it up into a left hand portion between 0 and 0.8, and a right hand portion from 0.8 to 1.0. If the first coin is Heads, I put it in the left portion, if tails in the right portion. If the second coin is Heads, I do the same on the expanded portion which

got used last time. I continue until all ten coin throws have been described. The result is an interval. For the case of ten throws with the first 8 being Heads, I actually find that the interval is from $0.96(0.8)^8$ to 0.8^8. In decimal this comes out as between 0.16106127 and 0.16777216 approximately. Now to transmit the result in arithmetic code, I simply choose a number in binary which is in this interval. If I receive such a number and know that there are only ten results and that the 0.8/0.2 partition has been used, I can deduce all the results. In fact the binary number 0.0010101 with only seven bits comes out as 0.1630625 exactly, which is inside the interval. So I can actually send seven bits to code the sequence- using the model m = 0.8 to do it. This involves a small amount of cheating which I leave you to ponder. Beating the ideal of 7.219 looks suspicious and is.

Some innocent experimenting with other models and a calculator may lead you to enlightenment faster than any amount of prohibited substances.

It is easy to see that the m = 0.8 model does better than any other. The reason is obvious: when we add up the logarithms of the reciprocals of the probabilities we are multiplying the improbabilities, and the minimum of this number is the maximum likelihood. So the maximal compression model in this case is the good old Maximum Likelihood model. It is not too hard to believe that in the case where a Bayesian convinces you that the MAP model is the best one for fitting the data that hasn't yet come in, then your best bet for compressing this data, if he is right, is the MAP model.

The overhead of sending the compression scheme, i.e. the model and the procedure for Arithmetic coding is, presumably, the same in both cases and may be ignored.

Next Up Previous Contents

Next: Compression for pdfs **Up:** Minimum Description Length Models **Previous:** Codes: Information theoretic preliminaries *Mike Alder*
9/19/1997

Compression for *pdf*s

Although the ideas are clear enough in the case of discrete distributions, it is not quite so easy to see how to use a gaussian *pdf*, for example, to compress a set of data. And yet in real life we always specify our real numbers to some precision, and the use of an infinite string of digits is a pretence. It simplifies some aspects of thinking about the world if we decline to say how many digits following the decimal point are going to be used, but it brings us some formidable complications in the language at a later date. (All the paradoxes of infinite sets, the existence of uncountable sets, arise from a desire to go beyond the confines of a finite language for essentially aesthetic reasons.) It is clear that to send one real number alone would require, for almost all real numbers, an infinite amount of information. So let us choose, from the beginning, some sensible bound on the precision of all the numbers we are going to transmit.

Now suppose we decide to work to one part in 2^P for some positive integer P. And suppose we are dealing with a set of points on the real line, in order to keep the example simple, and let us choose the five points {0.999, 0.099, 0.009, 0.001, 0.000}.

Suppose we decide to send these points to three decimal places; each place is a digit which takes four bits to code, since there are ten of them. So there are (5)(4)(3) = 60 bits. If we found a more efficient coding of the digits, we would be able to get $\log_2(10) = 3.32193$ bits per digit, giving 49.8289 as the total number of bits. Can we do better with an improved model which takes into account the fact that most of the density is near 0?

Suppose we define a $pdf,\ f(x)$, over the interval [0,1] by having the height of $f(x)$ as

$\frac{1}{3.7}$ for $x \in (0.1, 1.0]$,

$\frac{10}{3.7}$ for $x \in (0.01, 0.10]$,

$\frac{100}{3.7}$ for $x \in (0.001, 0.01]$,

$\frac{1000}{3.7}$ for $x \in [0, 0.001]$.

Then let us divide the unit interval up into the two parts from 0 to 0.1 and from 0.1 to 1, and for each point, write <.0??????> if the point is in the left tenth and <.1???????> if it is in the right 90%. The question marks have to be filled in by repeating the process a few times.

It is clear that repeating it three times to get <.000> locates the point as being in the interval from 0.00000.. to 0.001 (using decimal notation) so we have correctly specified the first three digits of 0.000. To write <.001> specifies a number in the interval from 0.001000... to 0.001999.. in decimal, which adequately specifies the first

three digits of 0.001.

We can specify then in binary the points 0.000 and 0.001 using the bit strings <.000> and <.001> respectively, thus getting two of the points in just 6 bits. The point 0.009 starts off in binary as <.001???> which tells us that it lies between 0.001 and 0.01. Since the *pdf* is uniform in this region, I simply code the interval in straight, equal interval, binary. I need to divide the region into 10 intervals to get 0.009. This can be done with an extra four bits, with a little wastage, Thus 0.009 is coded as <.0011001>. So 7 bits suffices for the third point out from the origin.

0.099 is the next point, and we start off with <.01???> to say it lies within the interval from 0.01 to 0.1. This has to be divided uniformly into 100 intervals which can be done using an extra 7 bits, again using equal interval binary, giving a total of 9 bits for this point.

Finally we have to code 0.999 and this starts off <.1????> to tell us it lies in the interval from 0.1000.. to 1.0000... If we divide this interval into 1000 intervals we can specify the location to the required precision with an extra 10 bits. This gives 11 bits for the last number. The total number of bits is thus 3 + 3 + 7 + 9 + 11 = 33 bits. I have used a sneaky convention to cut down on the bit lengths, and a pessimist might throw in another few bits, but this is a saving of over 13 bits in anybody's language.

If we consider the uniform probability distribution, recalling that we quantised to a precision of three figures, we see that the uniform distribution would give each of the five points a probability of $\frac{1}{1000}$ and so the information is $5\log_2(1000)$ which is the 49.8289 of the starting point. Using the assumed *pdf* we get that for the point <.000> for example, the probability is the height of the function multiplied by the width, which I shall generously put at $\frac{1}{2000}$, and the improbability is the reciprocal of this which is $(3.7)(\frac{2000}{1000}) = 7.4$, so the surprise or information content is $\log_2(7.4) = 2.8875$. We get the same for the string <.001> while for the point at 0.009 we get $\log_2(74) = 6.20945$. These correspond to the 3 bit and 7 bit codings of the first three points. Continuing we get:

$$2\log_2(7.4) + \log_2(74) + \log_2(740) + \log_2(7400) = 31.48167$$

which is a little better than we can realise with our coding scheme but shows the principle at work.

This, of course, all assumes that the overhead is negligible, which is hardly reasonable. On the other hand, the overhead can be specified by giving the *pdf*, and in transmitting the data in any other system there is *some* convention of coding employed. If I send it all in base 10, I ought to send also a translation of the ten digits in binary and the convention for using decimals if I want to compare like with like. I need to say something about the intended precision and the number of data points, too, so you will not need separators in the code to tell you when one stops and another starts. These issues apply to both the uniform coding and the non-uniform coding, so neglecting the overhead is unreasonable only if you take the view that there is a unique way to code real numbers and it presupposes a uniform density function.

It is plain, on a little reflection, that sending a list of N real numbers in the interval [0,1] by a process such as the above to a precision of 2^{-P}, for positive integers N and P, will require, if we use the uniform *pdf*, NP bits. If we have a $pdf,\ f(x)$, defined on the interval and replace it with an approximation uniform on intervals of size

2^{-P}, and if we suppose that the model is a good representation of the distribution of the data, then in any interval on the number x, the fraction of data points in that interval will be $Nf(x)(2^{-P})$ and they will be able to be coded using $\log_2(\frac{1}{f(x)(2^{-P})})$ bits each, i.e. $P + \log_2(\frac{1}{f(x)})$ each, so the total number of bits will be

$$\sum_{i=1}^{2^P} \frac{N}{2^P} f(i)(P + \log_2(\frac{1}{f(i)}))$$

Since $\sum_{i=1}^{2^P} f(i) = 2^P$ this simplifies to:

$$NP + N \sum_{i=1}^{2^P} f(i) log_2(\frac{1}{f(i)}) \frac{1}{2^P}$$

which in the limit for a finer and finer approximation to an integrable function f is:

$$NP + N \int_0^1 f(x) \log_2(\frac{1}{f(x)}) dx$$

If we compute $\int_0^1 f(x) \log_2(\frac{1}{f(x)}) dx$ for the *pdf* of the example, we get :

$$\left(\frac{1000}{3.7}\frac{1}{1000}\log_2(\frac{3.7}{1000})\right) + \left(\frac{100}{3.7}\frac{9}{1000}\log_2(\frac{3.7}{100})\right) + \cdots + \left(\frac{1}{3.7}\frac{9}{10}\log_2(\frac{3.7}{1})\right)$$

which comes out as

-2.183313 -1.156945 - 0.348909 + 0.4591278 = -3.2300396

which multiplied by the number of points, 5, gives -16.1502, a fair approximation to the number of bits saved by our choice of *pdf*. Purists might say I have been overly generous to myself here and there, but I merely wish to illustrate the principle.

The formula

$$NP + N \int_0^1 f(x) \log_2(\frac{1}{f(x)}) dx$$

or equivalently

$$NP - N \int_0^1 f(x) \log_2(f(x)) dx$$

is of some use in estimating potential savings. It should be noted that all the savings come from the case where the value of $f(x)$ is greater than 1, since here the contribution is negative so there is a saving. It is essential to

remember that this is the minimum possible bit length for representing the data *on the assumption that the model is the best possible probabilistic model for the data.*

Everyone will, of course, recognise the expression for the integral as defining the *entropy* of the $pdf,\ f$. If we use natural logarithms instead of logarithms to base 2, we merely change the units of measurement from bits to *nats* (or for some, less serious writers, *nits*).

If instead of working over the interval [0,1] we were to work over, say, [-1,1], we would need an extra bit to determine the sign of the number. On the other hand, if we are prepared to work to the same *relative* precision, we can give up a bit on the location, since knowing where things are on a region of twice the size means we can go to fewer binary places. It really rather depends therefore on what we mean by the precision of specification of a point. This in turn hangs rather on why we want to send a data set in the first place.

If you ask me to measure the diameter of some coins and to send you the results, I might give results in centimetres or inches, and it is likely to matter which. So the choice of a unit is vital and I have to send this information somehow. If you received a value in centimetres of 1.823456789, it is idle to pretend that you are going to care equally about the different digits. You will care, perhaps about the first digit, the 1, more than the second digit, unless as a result of having some crude measurements yourself you have already guessed the first will be 1. And you are likely to care less and less about succeeding digits until you don't care at all about the digits past some point. In fact you probably wouldn't believe them. If you discovered I had been using interferometric methods to get the eighth and ninth decimal places, you might think I was pretty daft, because those extra decimal places cost money and effort to obtain and normally do not carry a good deal of information, in the sense that you probably don't care too much about the values. Let us assume that you really do need the answers correct to one part in 2^P. If some third party were to choose units which were half the size of mine, he would be producing answers which were numerically twice as large, and in order to give the same precision relatively, he would need one less binary place than me. My interval of points will be, say between 0 and 1, his will be between 0 and 2, and if our intervals are divided up into 2^P subintervals, his will be twice as big as mine to convey the same amount of information. This has to be taken into account when specifying a *pdf*,*f*, because part of the specification is the domain space.

Suppose I consider a set of points given by a triangular *pdf* on the interval [0,1],

$$f(x) = 4x \ if \ 0 \leq x \leq 0.5; f(x) = 4 - 4x \ if \ 0.5 \leq x \leq 1$$

Suppose we choose to specify *N* points each to *P* binary places. I shall measure the information in *nats* to make the integration simpler, where 1 bit = $\ln(2) \approx 0.6931472$ nats

Then the best possible compression on this model is

$$NP\ln(2) - N\int_0^{\frac{1}{2}} 4x\ln(4x)dx - N\int_{\frac{1}{2}}^1 (4-4x)\ln(4-4x)dx$$

nats.

This becomes

$$NP\ln(2) - N\frac{2}{4}\left[\frac{u^2}{2}\ln(u) - \frac{u^2}{4}\right]_{u=0}^{u=2}$$

which is $NP\ln(2) + \frac{1}{2}N(1 - 2\ln(2))$ nats. Which is a rather small saving.

Now suppose the measurement units are changed so that the interval is halved. The required precision to specify the data is increased to $N(P+1)\ln(2)$ nats. The function *f*(*x*) now becomes

$$f(x) = 16x \; if \; 0 \leq x \leq 0.25; f(x) = 8 - 16x \; if \; 0.25 \leq x \leq 0.5$$

This still has to have integral 1 over the interval which is half the size. The total saving then comes out to be $N(P+1)\ln(2) + N\frac{1}{2}(1 - 2\ln(4))$ nats. So we have lost, for every point, one bit on the first term, but gained it on the second, giving no net change. It is easy to persuade yourself by a generalisation of this reasoning, that linear scaling does not change the compression resulting from using the model *f*, which is moderately comforting.

It follows that for a model defined over any bounded interval [*a*,*b*] of $\mathbb{R}$, it is safe to use the formula

$$NP + N\int_0^1 f(x)\log_2(\frac{1}{f(x)})dx$$

as an estimate of a bound on the compression attainable by using model *f*, provided we simply change the limits of the integral and bear in mind the implicit conditions on the precision with which we expect to specify the data points.

Intuitively, one would expect that there would always *be* a saving, that the worst case is when the function *f* is constant when the saving is clearly zero. In other words, reflecting on the function *f* as distorting the density of points in the space and on coding this information suitably as a way of expressing the data more efficiently, leads one to the belief that $\int_a^b f(x)\log(f(x))dx$ will be non-negative for any function *f* such that

$\int_a^b f(x)dx = 1$, and for any logarithm base. This is indeed the case, and I shall now prove it.

Proposition 12797

If data is generated in a bounded interval with density given by a *pdf f* and if the *pdf* is used to code the data, then for sufficiently large amounts of data *f* may be used to compress it except when the distribution is uniform.

Proof By the earlier observations, without loss of generality, we may suppose the interval is [0,1]. First, if the interval [0,1] is divided up into M equal subintervals and used to represent a finite probability distribution $\{p(i) : 1 \leq i \leq M\}$ it is easy to see that the expression $-\sum_{i=1}^{M} p(i)\log p(i)$ is maximised when

$\forall i, p(i) = \frac{1}{M}$. It has a value in this case of $\log(M)$. It may be minimised by making any one of the $p(i)$ 1 and the rest zero, when the minimum value of 0 is attained. If we transfer this to a function f on the unit interval, we have the same result for $f(i)\frac{1}{M} = p(i)$. Thus the value of

$\sum p(i)\log(p(i))$ lies between -*log*(*M*) and 0:

$$-\log(M) \leq \sum_{i=1}^{M} p(i)\log p(i) \leq 0$$

and since we can model this by the piecewise constant function f with $p(i) = f(i)\frac{1}{M}$,

$$-\log(M) \leq \sum_{i=1}^{M} f(i)\log(\frac{f(i)}{M})\frac{1}{M} \leq 0$$

which is to say

$$-\log(M) \leq \sum_{i=1}^{M} f(i)\log(f(i))\frac{1}{M} - \sum_{i=1}^{M} f(i)\log(M)\frac{1}{M} \leq 0$$

which becomes

$$-\log(M) \leq \sum_{i=1}^{M}(f(i)\log(f(i)))\frac{1}{M} - \log(M) \leq 0$$

i.e.

$$0 \leq \sum_{i=1}^{M}(f(i)\log(f(i)))\frac{1}{M} \leq \log(M)$$

And in the limit, when we replace the sum by an integral and the $\frac{1}{M}$ by dx we get

$$0 \leq \int_0^1 f(x)\log(f(x))dx < \infty$$

The zero value is attained only for the uniform distribution. □

The cheerful, and some would say irresponsible, treatment of limits will give a severe case of the uglies to

analysts, who may divert themselves by showing at length how it should be done.

It is easy to see that the value of $\int_0^1 f(x)\log(f(x))dx$ can be made arbitrarily large by concentrating f at some point. The wonderful Dirac delta function would allow an infinite amount of compression, if it only existed. Of course, if we agree to consider finite precision in the specification of the data, it makes little sense to worry overmuch about such things as limits or Dirac delta functions, they are accidents of the language to which we have become committed for historical reasons that no longer apply. An obsession with adventitious aspects of the linguistic apparatus makes sense only for lunatic Platonists.

It is plain that you never get the total number of bits reduced below zero, but it nevertheless follows that *any pdf* on a bounded interval which is not constant does some good in allowing compression *of a data set described closely by the pdf*. Moreover, any reasonable scheme for compressing a set of real numbers in a bounded interval into binary strings, which makes sense before you know what the numbers are, must assign binary strings differentially to intervals and hence determines a *pdf*.

If we want to work over the whole real line, the situation is somewhat more complicated. For points that are a long way from the origin, the number of bits necessary to get just the integer part goes up without bound. This, of course, means that it makes sense to shift the origin to where the points are and store the shift; this amounts to subtracting off the centroid and this obviously gets you some compression, although it is less than clear what model this corresponds to.

It is easy to compute $\int_{-\infty}^{\infty} f(x)\ln(\frac{1}{f(x)})dx$ for the case where f(x) is the gaussian or normal function $g_{[0,\sigma]}(x)$ and obtain $\ln(\sigma\sqrt{2\pi e})$. This is negative when $\sigma\sqrt{2\pi e} < 1$, i.e. when $\sigma < \frac{1}{\sqrt{2\pi e}}$. The conclusion is that for fat gaussians there is no advantage to be gained over just sending the data, and a fat gaussian is one where $\sigma > \frac{1}{\sqrt{2\pi e}} \approx 0.2419707$. This is quite different from the case where the *pdf* has compact support. It is easy to persuade yourself that the trouble is that unbounded intervals provide you, for any given size of data set, some residual probability of occurrence of a datum that nobody would believe for a minute belongs in with the rest of the data set. Pragmatic treatments of outliers has almost always been to pretend they didn't occur, but rationales for this conduct tend to lack something.

A gaussian *pdf* for the height of adult males fits the data quite well over several standard deviations, but the total number of male adults of negative height will always be zero, no matter how large the population. The reduction of the overhead in specifying *f* in a simple formula will have a cost in compression.

In practice, one can cheat quite sensibly by putting the same number of bits available for coding every data point past some distance from the origin, which means that for points which are very far

from the origin, there may be too few bits to specify them at all- a sort of overflow condition.

This is equivalent to making a decision to simply ignore some data if it is simply too far from the rest. Alternatively, I can decide on the precision I need by looking at the biggest distance from the origin of any of the data points, or I can have different precisions for different points and in this case I need a *pdf* on $\mathbb{R}$ which falls off as the number of bits needed to specify the integer part. This may be done rather neatly: See Rissanen's *universal pdf* on $\mathbb{R}$.

The fact that we get a sensible scale invariance for finite precision on compact intervals and no such result when the precision is allowed to become infinite and the domain of the *pdf* unbounded, suggests that the simplifications which we introduced into our lives many years ago in order to have very general theorems (of a rather impractical sort) may have been a horrible mistake. Maybe we should never have messed with infinite sets at all. Certainly we might argue for finite precision results, since nobody except a lunatic or a platonist philosopher thinks he is going to measure anything *exactly* with a ruler or any other sort of meter. And there is something metaphysical about unbounded *pdf*s in that no finite amount of data is going to demand them, and finite data is all we'll ever get.

My solution to the problem is going to be to choose some number N of standard deviations of a gaussian and use it to compress data within N standard deviations of the centre. If I ever get a datum outside my limit of N standard deviations, I shall simply ignore it. I shall choose N so that I get some compression from using the gaussian. How much compression I require will depend on how sober I am and other such matters contingent on the problem details.

This is $\mathcal{SIN}$ to platonist philosophers, and may give them conniption fits, which is good for them and stimulates their livers.

The extension to higher dimensions than one is essentially straightforward; in dimension *n* you need to send *n* numbers instead of just 1. The reader is invited to decide whether it is ever worth storing multivariate gaussians. Note that provided the entropy of the *pdf* is negative, then there is some number of points at which the saving beats the overhead on sending the *pdf*. Which we have not, so far, considered.

Next Up Previous Contents

Next: Summary of Rissanen Complexity **Up:** Minimum Description Length Models **Previous:** Compression for coin models *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Summary of the chapter **Up:** Minimum Description Length Models **Previous:** Compression for pdfs

Summary of Rissanen Complexity

Suppose we have two models for some set of data; we may think of the data as coming in sequentially for our purposes, in which case we can imagine the two models, q_1 and q_2, as predictors. The amount of information, which I called the *surprise*, which the data supplies to each model, is simply the sum over the data set of the logarithm of the improbability of each datum, as assessed by each model. Then all other things being equal, the less surprise or information supplied by the data to the model, the better the model. In the extreme case where one model is quite certain of its predictions and always correct, the data supplies no information at all. In the other extreme case the information supplied on each occasion is the logarithm of the number of possible states the datum point can occupy.

We suppose that there is some genuine random process specified by a pdf, p operating to produce the data, and that the model we have is represented by q. The sum $\sum_{\mathbf{x}\in D} p(\mathbf{x}) \log_2(\frac{1}{q(\mathbf{x})})$ gives the idealised value of the information supplied to the model q by data generated in fact by process p. This is larger than $\sum_{\mathbf{x}\in D} p(x) \log_2(\frac{1}{p(x)})$ the idealised value of the information supplied to the model by data generated from the model, the entropy of p, except in the case when $p = q$. The difference,

$$\sum_{\mathbf{x}\in D} p(\mathbf{x}) \log_2(\frac{1}{q(\mathbf{x})}) - \sum_{\mathbf{x}\in D} p(x) \log_2(\frac{1}{p(x)}) = \sum_{\mathbf{x}\in D} p(\mathbf{x}) \log_2 \left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right)$$

known as the *Kullback-Leibler* distance, is therefore always non-negative, and is used as a natural measure of the goodness of fit of one *pdf* to another. It is not symmetric. In our case we are obliged to estimate it since we do not know p, and we can estimate it only up to the unknown entropy of the actual distribution p. At least this allows us to compare two *pdf*s q_1 and q_2 on a given data set to see which is better.

It is fairly easy to see that if the two models q_1 and q_2 were predicting repeatedly from a finite list of events, and if they were to gamble with each other by making bets based on their separate predictions, using their probabilities to establish the odds for the bet and the stake size in any one of a number of sensible ways, then the better predictor, in the information sense, will make money from the worse one. The result in practical terms is simply to maximise the log-likelihood of the data for each model. If instead of two models we have a manifold of models, we simply maximise the log-likelihood over the manifold.

If other things are not equal, we may choose to penalise a model which has more parameters than another by some rate of exchange between the mean information supplied to the model by the data per data point, and the number of parameters in the model. The essential reason for this is our desire to use the model to predict what is going to happen next, and our confidence in the ability of our model to extract information from the data set maximally, while not being unduly affected by noise, is at issue. Occams Razor is the doctrine `Entities are not to be multiplied unnecessarily', or more popularly: `Given two theories, both of which account for the data, choose the simpler'. Many centuries after William of Occam, we may be in a position to say it in algebra with some precision instead of in Latin.

Why *should* we penalise a model with more parameters? One's intuitions, if one has messed around with models and data to any serious extent, should tell one that estimating too many parameters from too little data is unlikely to give anything very effective at predicting what is going to happen next. It seems reasonable that this state of affairs should be formulated as a penalty for number of parameters, the only question that remains to be solved is the rate of exchange between log likelihood (which will almost always improve for more parameters) and the number of parameters themselves.

The ideal rate of exchange would be determined by testing the model on later data to see how well it performs. After all, that is what we want models for. The measure of better performance ought to be the same as the measure above; the amount of information supplied to the model by later data, generated by the same process that gave the original data, will be less for the better model. What grounds have we for believing that the model with fewer parameters will do rather better than expected, while the model with more parameters will do rather worse? The model with too many parameters will be trying to model the randomness of any particular data set as if it were significant, when it isn't. It has actually extracted too much information from the data, finding pattern and structure when there isn't any. This is easily done; brains do it when they see butterflies or body parts in Rorschach blots, faces in flames and so on. This line of thought can lead to the *Akaike Information Criterion*. Here, we ask the question, if the data is a sample, D, generated by a *pdf p* and if we model it by a *pdf q*, what is the expected log-likelihood of the data relative to *q*? It turns out that in many cases the maximum log-likelihood choice of *q* is a biased estimate of the mean expected log-likelihood of *p* itself, and subtracting off the number of parameters removes the bias. See Akaike and also Sakamoto, *et al.* for an elucidation of these somewhat technical points. See also Shibata.

Alternatively, we can argue that since we know nothing of the model class which actually generated the data, we should abandon this attempt and go to more robust considerations. We can argue that compression is pertinent.

A *pdf* over $\mathbb{R}^n$ can be regarded as a method of compressing a set of points in $\mathbb{R}^n$. In doing this we simply think of the *pdf* as telling us how much to stretch bits of the space or alternatively to re-measure it so that we can code points efficiently, just as with Shannon coding. The amount of compression is greatest when the *pdf* is a good model of the distribution of the set. We can turn this on its head and define a `good model' as one which gives good compression. Certainly for two different models of the same family, the better model is the one with greater log likelihood. The `same family' of models means that there is some finite set of parameters which specify the model within the family, which is to say the `same family' of models comprise a manifold. So far, then, this is merely a new way of writing Maximum Likelihood models. But in the case where the class is larger than this and comprises, say, several manifolds with different dimension, we can also code the model parameters, and store the extra

overhead. This allows us to compute the total length of the data as coded using a model, plus the overhead of specifying the model. It is not particularly difficult to compute, for such things as gaussian mixture models with different numbers of gaussians what the net saving is. There is an advantage in having a better model, offset by the drawback of having to count extra parameters. Penalising the number of parameters is an obvious thing to do, but Rissanen offers a coherent explanation of precisely how to do this. The Rissanen approach, under certain circumstances, agrees with the Akaike approach asymptotically. Some might find this comforting, others mysterious.

Other work has been done from a Bayesian point of view on attempting to put in an Occam's Razor, that is to say a device for penalising a large number of parameters or entities. The idea here is that we can penalise a model family of higher complexity, that is to say with more parameters, by observing that the normalising constant $\int p(x|m)p(m)dm$ will be bigger when the integration is over a higher

dimensional manifold of models *m* thus reducing $p(m|x)$ in comparison with some alternative and `smaller' such manifold. The fact that the integral depends on the parametrisation of the manifold of models is grounds for some degree of nervousness among the innocent, but one can be persuaded this is not fatal. The result is that we favour a model from a class when the particular model accounts for the data well, when we have no strong prior prejudice against the model, and when on average the other members of the model class do fairly badly, possibly because there are so many of them. This has a certain charm. See MacKay's Doctoral Thesis in the bibliography for a well written propaganda job and some good references.

In the case of gaussian mixture modelling, the method implicitly recommended in the next chapter for dealing with pattern recognition problems if we opt for statistical methods, the choice of how many gaussians to use can be solved using the Akaike Information Criterion (AIC), the MDL or stochastic complexity criterion of Rissanen, or others, notably the BIC. Experiments, conducted on data sets actually generated by a known number of gaussians, suggest that the Rissannen criterion works tolerably well over a range of dimensions and numbers of gaussians, and performs better than the AIC. A rough guess of a sensible number may often be made by eyeballing the data under projection provided the dimension is not too high, and more sophisticated methods of automating the eyeballing to some extent will occur to the reasonably ingenious. Then some experiments with different numbers of gaussians and some arithmetic along the lines indicated above can yield a sensible number of gaussians to use in the model.

I shall return to the matter of compression and optimal modelling when we come to Neural Net models, which are notorious for having enormous dimensions for representing the data and consequently large numbers of parameters.

Next Up Previous Contents

Next: Summary of the chapter **Up:** Minimum Description Length Models **Previous:** Compression for pdfs *Mike Alder*
9/19/1997

Summary of the chapter

The basic assumption behind the probabilistic analysis of data is that there is some process or processes operating with the property that even when the input to the process is more or less known, the outputs differ. Rather like the drunk dart throwers of chapter one, the process *ought*, in a neat newtonian, causal, clockwork universe, to replicate its last output this time, but it generally does not. The classic examples, and the origins of probability theory, are in playing cards, throwing dice and tossing coins, where although we have a convincing causal model (Newtonian Dynamics) for the process, our ignorance of the initial states leaves us only `average' behaviour as effective data in deciding what to expect next.

The assumptions of probability theory are that when you replicate an experiment or repeat something like tossing a coin or throwing a die, what you actually do is to reapply a map with some generally different and unknown and indeed unknowable initial state. All we know about the initial states is the measure of those subsets of them that lead to different sets of outcomes.

We can simplify then to having a measure on the space of outcomes. From this model of random processes as non-random processes but with ignorance of the precise initial state, all of probability theory derives, and on this basis, much modern statistics is founded. We rapidly deduce the existence of probability density functions over the space of outcomes as measuring the results of repeating the application of the random variable an infinite (usually uncountably infinite) number of times. When there are only finitely many possible outcomes, we assign a probability distribution to the elementary outcomes. The practicalities of trying to decide whether one state of affairs is a repetition of another state of affairs are ignored on the grounds that it is too difficult to lay down rules. So what a replication *is*, nobody knows, or them as knows ain't saying. To highlight this point, consider a robot programmed with classical mechanics required to throw or catch a cricket ball or some similar mechanical task. It might be a programming problem, but the principles are clear enough. Now contrast this with the case of a robot which is intended to play a reasonable game of poker. To what extent is the information acquired in one game transferable to another? This is the problem of replication; it involves what psychologists call `transfer of learning', and we do not understand it. When we do, you will be able to play a poker game with two human beings and a robot, and the robot will catch you if you cheat. Don't hold your breath until it happens.

Model families, often parametrised as manifolds, spring out of the collective unconscious whenever a suitable data set is presented, and there are no algorithmic procedures for obtaining them. Again, this is bad news for the innocent, such as me, who wants to build a robot to do it. There are procedures for taking the manifold of models and choosing the point of it which best fits or describes the data. If there were only one we might feel some faith in it, but there are several. There is no consensus as to which procedure gives the best model, because there are competing ideas of what `best' means. There are three well known, and occasionally competing, procedures for picking, from some set of models, the best model for a given data set, and they all demand that someone, somehow, went and worked out what the

choice of models had to be. (Take a model, any model...). These three procedures, which we have discussed at some length, give rise to the Maximum Likelihood model, the Bayesian optimal model and the Minimum Description length model. Sometimes these are all different, sometimes not. Arguments to persuade you that one is better than another are crude, heuristic and possibly unconvincing. Tough luck, that's the way the subject is.

Thus we face some rather hairy problems. The state of the art is something like this:

We cannot in general evolve a model of any useful sort from the data as yet, we have to rely on people looking at the data, and then inspecting the entrails of chickens or gallivanting about in their subconsciouses or wherever, and bringing back a model family.

When we *have* a set of models described by finitely many parameters so that the models comprise a smooth manifold, there are several different ways of picking the best one from the set. We can compute a figure of merit for the pair consisting of a model and a set of data which the model might or might not have generated. This figure of merit is a real number called the *Likelihood*, and hence for a fixed data set there is a function defined from the manifold of models into $\mathbb{R}$. This function has a unique maximum often enough to make it tempting to use Maximum Likelihood as the natural meaning of the `best' model to account for the data. However, this pays no attention to the possible existence of other information which might predispose us to some other model; in particular, there might be a (prior) probability density function associated with the manifold of models. This, many people might feel in their bones, ought to be used in choosing the `best' model. Where the prior *pdf* comes from is a matter seldom discussed, but presumably it comes from some other source of data about the system under investigation: if we are into coin tossing then presumably it derives from having tossed other, different but similar, coins in the past.

Finally, if we have feelings in our bones about information theory as the right place to found statistical reasoning, and if we also feel in our bones a preference for simple models rather than complicated ones, we may be able to fall back on Rissanen style arguments if we are lucky, but many statisticians don't accept Rissanen's ideas. Rissanen gives us a chance to reject models which give a high likelihood but seem to be too complex, and to prefer simpler models with a lower degree of likelihood for the data. I discussed philosophical issues, I drew morals and extracted, some would say extorted, principles. I then went on to *pdf*s, starting with a coin model, and showing how the ML model for the coin compressed the results of tossing it. Then I compressed a set of points in the unit interval, using a *pdf* over [0,1]. Various useful and intuitive results were proved in a manner that no modern analyst would tolerate but that was good enough for Gauss.

The subjectivity which allows Statisticians to spend uncommon amounts of time in disputation means that choosing the answer that pleases you most is what is commonly done.

One should not be deterred from considering a theory just because of the air of dottiness about it which appears once one has stripped away the technicalities and described it in plain English. Theories are to be judged by their consequences, and on that criterion Probability Theory has been extremely successful. There are, nevertheless, quite serious problems with the semantics of the theory. For a start, the repetitions of the application of the *rv* mean that the different initial states have to be equally likely, but this is part of what we are trying to define by the apparatus of random variables. In applications, Rissanen has pointed out that there is no operational procedure for deciding if something is `random', and experts argue over the legitimacy of various techniques while the trusting just use them. Formalisation of

these ideas via the Kolmogorov axioms has meant that while the Mathematics may be impeccable, it isn't at all clear how reality gets into the picture. The innocent engineer who wants recipes not rationales can be a party to his own deception and often has been.

You can see why so many people dislike Stats. Just as you've finished learning some complicated methods of coping with uncertainty and doubt and finally come to believe you've got a stranglehold on the stuff, it leers at you and tells you it's all a matter of opinion as to whether it's OK to use them. For the innocent wanting certainty, it isn't as good a buy as religion.

Next Up Previous Contents

Next: Exercises **Up:** Statistical Ideas **Previous:** Summary of Rissanen Complexity *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Bibliography **Up:** Statistical Ideas **Previous:** Summary of the chapter

Exercises

These little exercises are of the simplest possible sort, and involve mostly tossing coins. They should therefore be easy for anyone who has even the smallest knowledge of probability theory. Try them on your friendly neighbourhood probabilist if you can't do them yourself.

1. You are asked by a friend to show him the coins you have in your pocket or purse and you find two ten cent coins. He selects one of them, and then proceeds to toss it ten times, obtaining eight heads and two tails in some order. He then offers to bet on the results of tossing the coin again. Feeling that the data is too restricted to allow you to estimate odds sensibly, you toss the other coin one thousand times (rather quickly) and keep count of the results. You get 503 Heads and 497 Tails, but you note that the variation if you partition them into groups of ten is considerable and accords well with the predictions of a binomial model with p(H) = 0.5. Present a careful argument to determine what odds you would offer your friend in a bet for a dollar on the next throw of his coin coming heads. Repeat for the case where the bet is one hundred dollars. Would it make a difference if the friend was called Persi? If you were allowed to be the one to toss the coin? Could the order of the results make a difference, for example if the two tails came first? Last? What about the possibility that the number of throws was chosen not because ten is a nice round number but because your friend threw two tails at the end because he was getting tired and thought that two bad throws meant it was time to stop?
2. The same situation as in the last exercise occurs, but this time you have one ten cent coin and one twenty cent coin. Your friend takes the ten and you do your experiment with the twenty. Do you feel that this makes no substantive difference to the process of throwing your coin one thousand times and applying the result to expectations about *his* coin? Would there be some measure of relevance which you were using implicitly? How about if there was one ten cent coin and (a) a saucer (b) a ten dollar bill and (c) a small piece of green cheese, in your pocket? Would you feel that the relevance of tossing the ten dollar bill in the air so small as to be useless, but a twenty cent coin might give you some information? If so, explain how to program a robot to calculate the relevance. What would you do with the green cheese?
3. Suppose a martian or some visitor from a distant planet, or maybe an intelligent octopus, were to be watching you and your friend, and suppose the kibitzer had never seen a coin tossed before or anything remotely like it. Would it, do you feel, be inclined to feel that tossing the second coin or possibly the ten dollar bill might be a good idea? Could it conceivably examine the dynamics of the processes and make any inferences as to relevance? If any of the parties present were a subjective Bayesian, would he have have any explanation of where the kibitzer got his, her or its

prior from before it did the experiment with a different coin?

4.

You are given 2000 points in the unit interval to a precision of 10 bits. There is thus quite a good chance that several of the points will coincide, up to the given precision. Suppose they have in fact been generated by a uniform distribution; can one expect, on average, to compress them by rather better than by sending 20,000 bits, how might one do this, and ought one to do it?

5.

You work to one bit precision on the unit interval and you have three data points, $\{0, 0, 1\}$ .You wish to find a good model for the data. The maximum likelihood model says that it's twice as likely to be a as a 1. You note however that with three data points the only possibilities are three 0s, two 0s and a 1, two 1s and a 0, and three 1s. Since you wish to be able to get good predictions on the next 1000 points, it seems absurd to restrict yourself to one of a class of four models, the only ones maximum likelihood *could* provide you with, if you assume a binomial model family to start from and have only three data. You therefore use bayesian methods and assume a uniform prior on the grounds that it contains least prejudice about the way things are. Try the calculation and see how much good it does you when you take the MAP estimate. Are there any grounds for a prior prejudice in favour of a binomial model with equal probabilities for the two outcomes? If the data came from tossing a coin and assigning a zero to Heads and a 1 to Tails, would you alter your answer to the last part? If the bits represented polarisation of a signal from outer space, would you feel differently about things? Do you think learning a bit of Physics might help in the last case? Just how ignorant *are* you?

6.

Suppose that in the last example, the data comes in sequentially after the first three arrive in a lump, and that we proceed by updating our estimate of the posterior distribution continuously. This procedure amounts to getting a sequence of points in the space of posterior distributions, which bears a passing resemblance to the unit interval. If the data were in fact generated by a binomial process with probability of getting 0 equal to *p*, there ought to be some grounds for thinking that the sequence will almost surely converge to this value. By formalising what the terms mean, prove that this is indeed the case.

7.

With the assumptions of the last question, how many points are needed in order to justify the use of the appropriate binomial model as a compression device?

8.

If the sequence in fact consists of the bit sequence consisting of (0,0,1) repeated indefinitely, how many points are required before the repetition model beats the binomial model? How many would *you* need?

9.

Suppose that you have a gaussian mixture model of some data, where the data was in fact generated by simulating k gaussians in the plane. Are there circumstances in which you would expect a really effective method for penalising too many coefficients to (a) underestimate or (b) overestimate k? In other words, could underestimating (overestimating) k ever be the sensible

thing to do?

Next Up Previous Contents

Next: Bibliography **Up:** Statistical Ideas **Previous:** Summary of the chapter *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Decisions: Statistical methods **Up:** Statistical Ideas **Previous:** Exercises

Bibliography

1. Albert Morehead and Geoffrey Mott-Smith, *Hoyles Rules of Games* New American Library, 1983.
2. Jorma Rissanen, *Stochastic Complexity in Statistical Inquiry* World Scientific Pub.Co. 1989.
3. Jorma Rissanen, *Stochastic Complexity* J.Roy.Statist.Soc.**B** (1987)**49**, No. 3, pp223-239 and 252-265.
4. Jorma Rissanen, *Stochastic Complexity and the Maximum Entropy Principle*, Jorma Rissanen, IBM Almaden Research Center, K52/802, San Jose, Ca. 95120-6099, USA.
5. Peter Walley, *Statistical Reasoning with Imprecise Probabilities*, Chapman and Hall, 1991.
6. E.T. Jaynes, *Papers on Probability, Statistics, and Statistical Physics* (Ed. R.D. Rosenkrantz. Kluwer Boston, 1983.
7. E.T. Jaynes *Probability Theory: The Logic Of Science* ftp site : bayes.wustl.edu on subdirectory *Jaynes.book*.
8. R. Thom, *Structural Stability and Morphogenesis* Translated from the French ed., as updated by the author, by D. H. Fowler. W. A. Benjamin, 1975.
9. Ralph Abraham, *Foundations of Mechanics* Benjamin, 1967.
10. Gregory Chaitin, *Information, Randomness and Incompleteness* World Scientific 1987.
11. Gregory Chaitin, *Randomness and Mathematical Proof*, **Scientific American**, 232 pp 47-52, May 1975.
12. Irving John Good, *Good thinking : the foundations of probability and its applications* University of Minnesota Press, 1983.

13.

Irving John Good, (Ed.) *The scientist speculates / an anthology of partly-baked ideas.* London : Heinemann, 1962. Here for fun.

14.

Irving John Good, *The estimation of probabilities : an essay on modern Bayesian methods* Cambridge, Mass : M.I.T. Press, 1965.

15.

Claude Shannon *A Mathematical Theory of Communication* Bell Systems Technical Journal, **47** pp143-157.

16.

Claude Shannon and W. Weaver *The Mathematical Theory of Communication*, Univ. of Illinois Press, 1949.

17.

Thomas Cover, Joy Thomas *Elements of Information Theory* Wiley 1991.

18.

Y.Sakamoto, M Ishiguro and G. Kitagawa *Akaike Information Statistics* D. Reidel 1986.

19.

H. Akaike *A new look at the statistical model identification* IEEE Trans. Aut. Control. **19** pp716-723 1974.

20.

G. Schwartz *Estimating the dimension of a model* Ann. Stat. **6** 2, pp461-464 1978.

21.

R. Shibata *Asymptotically efficient selection of the order of the model for estimating parameters of a linear process.* Ann. Stats. **8** pp147-167 1980.

22.

David J.C. MacKay *bayesian Methods for Adaptive Models* (Doctoral Thesis from CalTech) 1992.

23.

T.P. Speed and Bin Yu *Model Selection and Prediction: Normal Regression* Ann. Inst. Staist. Math. **45** No.1, pp35-54. 1993.

Next Up Previous Contents

Next: Decisions: Statistical methods **Up:** Statistical Ideas **Previous:** Exercises *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: The view into **Up:** An Introduction to Pattern **Previous:** Bibliography

Decisions: Statistical methods

In the last chapter I took a somewhat jaundiced look at the raw ideas of modern statistics and probability theory; I can only hope that the thoughtful reader is still with us and has not abandoned the entire enterprise in disgust.

In that chapter, I drank deep of Philosophical matters to an extent which can drive many an engineer bananas. Some people like that kind of thing, but it doesn't go with the hard headed attitude of those who want to get on with programming their robot to distinguish between eggshells and the best china. The trouble is, that it is possible to sell a kind of intellectual snake oil to such engineers, as the existence of all those books on Fuzzy Set Theory makes clear, so those engineers should start getting more sceptical. Looking critically at the relatively honest and only faintly crackpot ideas of the probabilists is a good start. What looks like a promising avenue of enquiry to some can look to others like the ravings of a glue-sniffing, born-again fruit bat. While, in the fullness of time, truth will out and either your programs will work or they won't, it can take a long time to get to this happy state, and some critical thought about what kind of methods merit consideration can save a lot of effort.

In other words, I am only mildly apologetic. If that.

In this chapter I shall give some statistical methods for coping with the situation where some measurements have been made, the data consists of some collection of points in $\mathbb{R}^n$,each point labelled as belonging to some finite set of categories, and the problem is to decide to which category a new, as yet unlabelled, point belongs. If the attempt looks to be a desperate one, fraught with risk and uncertainty, and if the justifications offered seem to be shonky and unconvincing, we can only say with Bruce Fairbairn's famous character, `if you know a better 'ole, go to it.' The reader who still has his feet up and is getting slightly sloshed would do well to sit up straight and drink some coffee.

First an important principle in any branch of Science or Engineering: eyeball the data. Look at it, run it through your fingers, get a feel for it. Learn to know its little vagaries. In these days of instant packages which allow you to get out results using methods you don't understand, implemented by people you don't know, on equipment you haven't tested, it is very easy to produce totally mindless garbage, orders of magnitude faster and stupider than at any time in history. Have no part of this. Mindless stupidity is always popular with the mindlessly stupid, but it doesn't cut the mustard. So, I repeat, **EYEBALL THE DATA!** In the case of data in dimension 12, say, this is not so easy as if the dimension is 2, when you can plot O's and X's as I did in chapter one. But given the powers of the computer it is not much harder to project it from dimension *n* onto the screen of a computer, and to spin it and look at it from several angles. So we start off with a painless bit of linear algebra.

- The view into
- Computing *PDF*s: Gaussians
  - One Gaussian per cluster
    - Dimension 2
  - Lots of Gaussians: The EM algorithm
    - The EM algorithm for Gaussian Mixture Modelling
  - Other Possibilities
- Bayesian Decision
  - Cost Functions
  - Non-parametric Bayes Decisions
  - Other Metrics
- How many things in the mix?
  - Overhead
  - Example
  - The Akaike Information Criterion
  - Problems with EM
- Summary of Chapter
- Exercises
- Bibliography

Next Up Previous Contents

Next: The view into **Up:** An Introduction to Pattern **Previous:** Bibliography *Mike Alder 9/19/1997*

The view into $\mathbb{R}^n$

Suppose you have a few hundred vectors of two categories, each vector of dimension 12. It may be possible to look at the column of numbers and tell by eye which is which. If the points are well separated this can sometimes be done. This primitive and basic eyeballing of the data is well worth trying. Sometimes it can show that you have done something daft, such as having a zero vector in both categories, which has been known to be produced by a program bug.

If the number of categories is bigger than two, or the data less well separated, looking at the columns of numbers may be uninformative. But just as a photograph of, say, a tree is a projection of a three dimensional object onto two dimensions, so it is possible, indeed easy, to project from $\mathbb{R}^n$ down to $\mathbb{R}^2$ for any n. And to get from any bounded piece of $\mathbb{R}^2$ to the computer screen is just shrinking or stretching and shifting. Any view of a piece of $\mathbb{R}^3$ will leave somethings hidden behind others and leave it unclear as to the relative distances, but doing any rotation about almost any axis will show us the relative spacings and reveal what was hidden. If we can rotate $\mathbb{R}^n$ and look at the resulting data from different perspectives, we can sometimes see some useful structure in the data.

Suppose we have two orthogonal vectors $\mathbf{u}, \mathbf{v}$ in $\mathbb{R}^n$ of length 1. Then if $\mathbf{x}$ is a point in $\mathbb{R}^n$ and $\cdot$ is the usual dot product, the projection of $\mathbf{x}$ onto the plane spanned by $\mathbf{u}, \mathbf{v}$ is $(\mathbf{x} \cdot \mathbf{u})\mathbf{u} + (\mathbf{x} \cdot \mathbf{v})\mathbf{v}$. So by taking the two numbers $\mathbf{x} \cdot \mathbf{u}, \mathbf{x} \cdot \mathbf{v}$ we map the point $\mathbf{x}$ into $\mathbb{R}^2$. It is then straightforward to plot these points on the computer screen, and one of the programs on disk does this for you.

Rotations in $\mathbb{R}^n$ are most easily thought about as composites of rotations in a choice of plane. The three dimensional case is rather special, since in choosing a plane to rotate in $\mathbb{R}^3$ we are also choosing an orthogonal line which is fixed. In $\mathbb{R}^4$, we can leave a two dimensional subspace fixed, we rotate about a plane. So it is simpler to stop thinking in terms of axes of rotation and to concentrate on the plane that is moved into itself. It is easiest to take these to be defined by some choice of two different axes.

We therefore obtain a basic rotation in $\mathbb{R}^n$ by choosing distinct *i* and *j* between 1 and *n*. We write down

the identity *n* by *n* matrix, and change the locations (*i*,*i*), (*i*,*j*),(*j*,*i*) and (*j*,*j*). In the (*i*,*i*),(*j*,*j*) locations we write $cos(\theta)$, for some choice of θ ,in location (*i*,*j*) we write $sin(\theta)$ and in location (*j*,*i*) we write $-sin(\theta)$ for the same choice of θ. The new matrix rotates $\mathbb{R}^n$ by an angle θ in the (*i*,*j*) plane.

By making any choices of *i*,*j* and θ, we get different rotations. By composing one after another we can get any special orthogonal matrix.

Initialising the vectors $\mathbf{u}$ and $\mathbf{v}$ to being, say, the first two elements in the standard basis for $\mathbb{R}^n$, and then applying a sequence of rotations, we can view the points of the data set from a variety of angles. We regard our two `window vectors' as the things to be operated on rather than the data points, as this is a lot quicker.

There are some obvious modifications to the above algorithm, such as scaling (zooming in or out) and shifts in the space. After a large number of rotations, it may happen that the window vectors get deformed out of true: they may no longer be orthonormal as a result of the accumulation of rounding errors in the matrix multiplications. It is fairly easy to renormalise the window vectors if this happens.

There are several advantages to inspecting the data by this method; outliers may be found (and sometimes recognised as errors), and the decision as to whether to use Statistical or Neural Net methods or alternatives may be taken in some cases. It has been found, for example, that some data appear to consist of one gaussian for each category of point under every explored projection, and in this case it is tempting to automate the visual inspection: make random choices of rotation, for each projection test to see if the distribution is acceptably gaussian, using any of the standard statistical methods, and repeat. If it is gaussian from enough directions, where `enough' depends on *n*, we might feel that it would be sensible to use a gaussian distribution to model the data. If it looks very much like a collection of convex sets bounded by hyperplanes, one might use the classical neural nets.

On the disk may be found some images of vowels obtained by extracting from the NIST data base samples of many speakers saying `Ah' and `Oo'. Each sound was sampled and a Fast Fourier Transform applied to obtain the power spectrum. The results were binned into 12 different frequency bands, spaced more or less logarithmically along the frequency axis, and the resulting twelve numbers used to describe the sound. An utterance was thus turned into a trajectory in $\mathbb{R}^{12}$, although rather a short one in this case. The collection of such trajectories is fitted reasonably well by a twelve dimensional gaussian distribution, a different one for each vowel sound. Classifying vowels by these means may be accomplished by the methods of the next section but one. A projection program which allows you to look at the data is also available on the disk.

There is a program called *fview* written by Gareth Lee which allows projection of views and rotations of higher dimensional data. The program is available by anonymous FTP from ``ciips.ee.uwa.edu.au'' (I/P number 130.95.72.69), is located in directory ``pub/speech/tools/fview'', and is called ``fview1.0.tar.Z''. It takes up about 2.5Mbytes. It should run straight-out-of-the-box on SparcStations and does not require XView to be installed. It will need to be compiled to run on other machines and will need XView. The same site contains papers and data concerned with Pattern Recognition in various sub-directories.

After fview had been written, the UNIX utility XGOBI became public and is more useful for some kinds of data. It is a very nice piece of work, and is warmly recommended to everyone who works with data in dimensions higher than two.

I cannot stress too much the fundamental silliness of trying to solve a problem of pattern recognition using a package that saves anybody from ever looking at the data or thinking about how it was obtained or generated. Any method of examining the data by eye is a help. Trying to get out of the business of getting your hands dirty is to embrace intellectual death, you turn into one of those arty types who can talk about everything but can't actually *do* anything. This is even worse than being one of the hearty, unintellectual types who can remember the recipes but has no idea why they work or how anybody devised them.

Next Up Previous Contents

Next: Computing PDFs: Gaussians **Up:** Decisions: Statistical methods **Previous:** Decisions: Statistical methods *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: One Gaussian per cluster **Up:** Decisions: Statistical methods **Previous:** The view into

Computing *PDF*s: Gaussians

By way of light relief, this section will concentrate on the simple and practical problem of modelling the distribution of points using the multivariate gaussian function. I have already indicated how this can be used for making decisions as to what category a point belongs to, and I shall go into this matter in more detail shortly. At present, let us settle for an attempt to fit a good gaussian function to a set of data. Whether we see this as a form of data compression which replaces detailed knowledge of the data with some rather vaguer approximation thereto, or an estimate of some deeper underlying process responsible for having generated the data (by some ineffable mechanisms it is better not to eff) does not much matter. Either way we go through a certain computation. The terminology tends to suggest the estimation of some underlying ineffable whatnot, but it would be a mistake to suppose that the accompanying metaphysical baggage is either necessary or desirable, still less that the writer has a commitment to it.

To focus ideas, the reader might like to recall the problem of telling the guys from the gals by fitting gaussian distributions to the points on the height-weight diagram of Figure 1.2. back in chapter one, or of telling `aah' from `ooh' in twelve dimensions if he has had a look at the data of the disk or played with the fview program.

- One Gaussian per cluster
  - Dimension 2
- Lots of Gaussians: The EM algorithm
  - The EM algorithm for Gaussian Mixture Modelling
- Other Possibilities

Next Up Previous Contents

Next: One Gaussian per cluster **Up:** Decisions: Statistical methods **Previous:** The view into *Mike Alder 9/19/1997*

One Gaussian per cluster

Suppose I have a cluster of points in $\mathbb{R}^n$, say the heights and weights of a number of adult males, as in chapter one, *Fig. 1.2*, looking at only the Xs. In low dimensions such as this we can look at them and say to ourselves `yes, that can be modelled by a gaussian.' Well *of course* it can. Any set of points can be. Let us remind ourselves of the elementary methods of doing so.

Let

$$\mathcal{X} = \left\{ \begin{pmatrix} x_1^1 \\ x_1^2 \\ \vdots \\ x_1^n \end{pmatrix}, \begin{pmatrix} x_2^1 \\ x_2^2 \\ \vdots \\ x_2^n \end{pmatrix}, \cdots, \begin{pmatrix} x_M^1 \\ x_M^2 \\ \vdots \\ x_M^n \end{pmatrix} \right\}$$

be a set of *M* points in $\mathbb{R}^n$. The *centre* of the *M* points (or *centroid*, or *centre of gravity*), is the point

$$\mathbf{m} = \begin{pmatrix} \bar{x}^1 \\ \bar{x}^2 \\ \vdots \\ \bar{x}^n \end{pmatrix}$$

where $\bar{x}^i = \frac{1}{M} \sum_{j=1}^{j=M} x_j^i$, i.e. each component is the mean or average of the *M* values of that component obtained from the *M* points.

In vector notation we write simply:

$$\mathbf{m} = \frac{1}{M} \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x}$$

We can take it that the centroid of a set of points contains some first order approximation to saying something about the set in terms of a simple vector. To say more about the set we proceed to the second

order moments. These are the average values of the terms $(x^i - \bar{x}^i)(x^j - \bar{x}^j)$ in the given vectors, and there are n^2 of them, so we arrange them in an n by n matrix called the *covariance matrix*. Formally, the covariance matrix, $\mathbf{V}$ will have the entry in row i and column j given by

$$\mathbf{V}^i_j = \frac{1}{(M-1)} \sum_{r=1}^{M} (x^i_r - \bar{x}^i)(x^j_r - \bar{x}^j)$$

In vector notation this may be compactly written as

$$\mathbf{V} = \frac{1}{(M-1)} \sum_{\mathbf{x} \in \mathcal{X}} (\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T$$

The use of M-1, where the innocent might have expected M, occurs for reasons into which I shall not go; intuitively one can argue that in subtracting off the mean, $\mathbf{m}$, from every point, we have thrown away one point's worth of information. It is not unlike subtracting off a point from a set of M points in linear algebra in order to see if the original points lie on a line or plane; one looks at the M-1 non-zero points to see if they are linearly dependent. Here we have not subtracted off one of the points, we have subtracted off $\left(\frac{1}{M}\right)^{th}$ of each of them.

The matrix $\mathbf{V}$ is of course symmetric by construction, and (although this is not so immediately evident) positive semi-definite. Since it is symmetric, elementary linear algebra assures us that it may be diagonalised by a rotation matrix, i.e. there is an orthogonal matrix with determinant 1, $\mathbf{Q}$, and a diagonal matrix $\mathbf{\Lambda}$, so that $\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} = \mathbf{V}$. The diagonal elements of $\mathbf{\Lambda}$ are called the *eigenvalues* of $\mathbf{V}$, the image of the standard basis in $\mathbb{R}^n$ by $\mathbf{Q}$ is called the *eigenbasis* for $\mathbf{V}$ and the image of the basis vectors is called the set of *eigenvectors*. Traditionally, any multiple of an eigenvector is also an eigenvector, so when the books refer to an eigenvector they often mean a one dimensional eigenspace, and sometimes they just meant eigenspace.

$\mathbf{Q}$ is unique provided that the eigenvalues are all different, whereupon the eigenspaces are all one dimensional. Each eigenvalue is non-negative (an immediate consequence of the positive semi-definiteness). If the eigenvalues are all positive, then the matrix $\mathbf{V}$ is non-singular and is positive definite. It is again a trivial consequence of elementary linear algebra (and immediately apparent to the meanest intellect) that in this case $\mathbf{V}^{-1}$ is diagonalisable by the same matrix $\mathbf{Q}$ and has diagonal matrix the inverse of $\mathbf{\Lambda}$, which simply has the reciprocals of the eigenvalues in the corresponding

places. Moreover, there is a symmetric square root of $\mathbf{V}$, *conveniently called* $\mathbf{V}^{\frac{1}{2}}$, which can be diagonalised by the same matrix $\mathbf{Q}$ and has diagonal terms the square roots of the eigenvalues in the corresponding places. By a square root, I mean simply that $\mathbf{V}^{\frac{1}{2}}\mathbf{V}^{\frac{1}{2}} = \mathbf{V}$.

Having obtained from the original data the centre **m** and this matrix $\mathbf{V}$, I shall now define the function

$$g_{[\mathbf{m},\mathbf{V}]} : \mathbb{R}^n \longrightarrow \mathbb{R}$$

by

$$g_{[\mathbf{m},\mathbf{V}]}(\mathbf{x}) = \frac{1}{(\sqrt{2\pi})^n \left(\sqrt{det(\mathbf{V})}\right)} e^{-\frac{(\mathbf{x}-\mathbf{m})^T \mathbf{V}^{-1}(\mathbf{x}-\mathbf{m})}{2}}$$

and assert that this function is going to be my probabilistic model for the process which produced the data set, or alternatively my preferred device for representing the data compactly.

If a rationale for this choice rather than some other is required, and the enquiring mind might well ask for one, I offer several:

First, it is not too hard to prove that of all choices of $\mathbf{m}$ and $\mathbf{V}$, this choice gives the maximum likelihood for the original data. So given that I have a preference for gaussians, this particular gaussian would seem to be the best choice.

Second, I can argue that just as the centre $\mathbf{m}$ contains first order information telling us about the data, so $\mathbf{V}$ gives us second order information- the central moments of second order are being computed, up to a scalar multiple.

Third, it is easy to compute the vector $\mathbf{m}$ and the matrix $\mathbf{V}$.

Fourth, I may rationalise my preference for gaussians via the central limit theorem; this allows me to observe that quite a lot of data is produced by some process which aims at a single value and which is perturbed by `noise' so that a large number of small independent factors influence the final value by each adding some disturbance to the target value. In the case where the number of factors gets larger and larger and the additive influence of each one becomes smaller and smaller, we get a gaussian distribution. Of course, it is hard to see how such an assumption about the data could be verified directly, but many people find this argument comforting.

And finally, if it doesn't look to be doing a good job, it is possible to discover this fact and abandon the model class, which *is* comforting. In that event, I have other recourses of which you will learn more ere long.

These justifications are not likely to satisfy the committed Platonist philosopher, who will want to be persuaded that this choice is transcendentally right, or at least as close as can be got. But then, keeping Platonist philosophers happy is not my job.

In dimension one we can draw the graph of the function and the data set from which it was obtained by the above process. The covariance matrix $\mathbf{V}$ reduces to a single positive number, the *variance*, and its square root is usually called the *standard deviation* and written σ.

The points satisfying $(\mathbf{x}-\mathbf{m})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{m}) = 1$ are therefore the numbers $\mathbf{m} \pm \sigma$.

In dimension two, it is possible to draw the graph of the function, but also possible to sketch the sections, the curves of constant height. These are ellipses. In particular, the points $\mathbf{x}$ satisfying

$$(\mathbf{x}-\mathbf{m})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{m}) = 1$$

fall along an ellipse in the plane. A simple computation shows that the integral of the function over the interior of this ellipse has the value $1 - e^{-\frac{1}{2}}$ which is about 0.4. Thus almost 40% of the points

distributed according to a gaussian lie within one `standard deviation' of the centre in two dimensions. This contrasts with the one dimensional case of course, where it is nearer 67%.

In dimension *n*, the set

$$\{\mathbf{x} \in \mathbb{R}^n : (\mathbf{x}-\mathbf{m})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{m}) = 1\}$$

which might reasonably be called the set of points at one standard deviation, is a hyperellipsoid. This follows of course from $\mathbf{V}$ being positive definite. If any of the eigenvalues are zero, the hyperellipsoid is said to be *degenerate*. The proportion of the distribution within one standard deviation goes down as the dimension goes up.

It is easy enough for even the least imaginative of computer programmers to visualise a cluster of points sitting in three dimensions and a sort of squashed football sitting around them so as to enclose a reasonable percentage. Much past three, only the brave and the insane dare venture; into which category we fall I leave to you to determine.

- Dimension 2

Next Up Previous Contents

Next: Dimension 2 **Up:** Computing PDFs: Gaussians **Previous:** Computing PDFs: Gaussians *Mike Alder 9/19/1997*

Dimension 2

In dimension two, it is often convenient, in these days of nice computer graphics, to draw the ellipse and display the data and the ellipse on a screen. This can conveniently be accomplished by finding a parametric representation of the ellipse.

To see how this may be done, shift back to the origin by subtracting $\mathbf{m}$ from everything. We now have an ellipse given by

$$\{\mathbf{x} \in \mathbb{R}^2 : \mathbf{x}^T \mathbf{V}^{-1} \mathbf{x} = 1\}$$

Now suppose $\mathbf{V}^{-1}$ has a square root $\mathbf{A}$ which, like $\mathbf{V}$ and $\mathbf{V}^{-1}$ is symmetric. Then we can write the above equation as

$$\{\mathbf{x} \in \mathbb{R}^2 : \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} = 1\}$$

since $\mathbf{A}^T \mathbf{A} = \mathbf{A}\mathbf{A} = \mathbf{V}^{-1}$.

Now if $\mathbf{y} = \mathbf{A}\mathbf{x}$ we have $\mathbf{y}^T \mathbf{y} = 1$, which means that $\mathbf{y}$ is on the unit circle, S^1, in $\mathbb{R}^2$. We can therefore describe the ellipse as

$$\{\mathbf{x} \in \mathbb{R}^2 : \mathbf{y} = \mathbf{A}\mathbf{x} \ \& \ \mathbf{y} \in S^1\}$$

or

$$\{\mathbf{x} \in \mathbb{R}^2 : \mathbf{x} = \mathbf{A}^{-1}\mathbf{y} \ \& \ \mathbf{y} \in S^1\}$$

or

$$\left\{\mathbf{x} \in \mathbb{R}^2 : \exists \theta \in [0, 2\pi], \mathbf{x} = \mathbf{A}^{-1} \begin{pmatrix} cos(\theta) \\ sin(\theta) \end{pmatrix}\right\}$$

It is easy to draw sets on a computer when they have been parametrised by a single real number. The following few lines of C code indicate how we can trace the path of a circle with time running from 0 to 2π, or at least a discrete approximation to it:

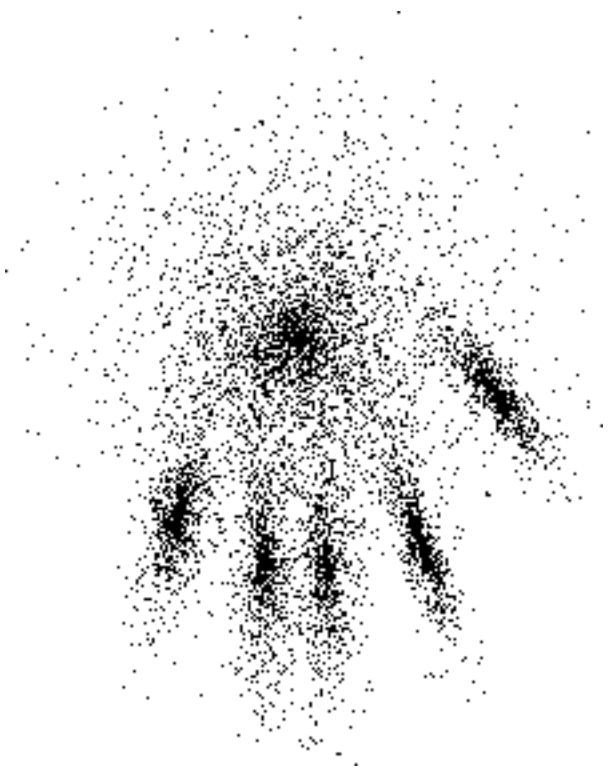
```
for(int_time = 0; int_time < 629; int_time++){

   time = int_time/100;
   x =  100* cos(time) + 200;
   y =  200 - 100*sin(time);
   putpixel(x,y);
};
```

This will draw a circle at centre (200,200) of radius 100. The choice of 629 calls for some explanation which I leave to the reader. The drawing ellipse, $\mathbf{A}^{-1}$ is simply $\mathbf{V}^{\frac{1}{2}}$ which we have already seen how to compute. This operates on the points of the unit circle to stretch them in the right way to draw a one standard deviation ellipse: if you want more than one standard deviation, the modification is straightforward.

The result of generating points according to a few gaussian distributions (well, more or less. It was faked, of course) in the plane and displaying them on a computer is shown in *Fig.4.1*. Six gaussians were chosen so as to produce a ghostly hand which may be seen with the eye of faith.

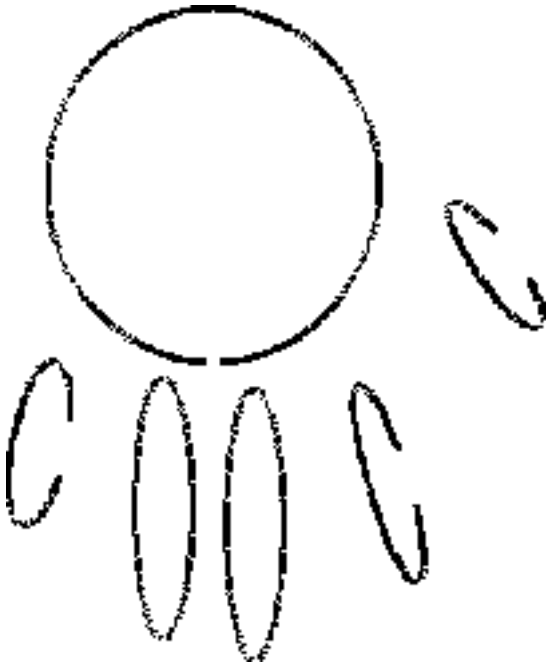
Figure 4.1: Simulated gaussians in $\mathbb{R}^2$.



The ellipses representing not one but 3 standard deviations are (mostly) drawn in the next diagram, *Fig.4.2*, where they should satisfy the most captious that they are doing something to represent the

distribution of the points in the data set all on their own.

Figure 4.2: Ellipses drawn at three standard deviations for the above data.



Next Up Previous Contents

Next: Lots of Gaussians: The **Up:** One Gaussian per cluster **Previous:** One Gaussian per cluster *Mike Alder*
9/19/1997

Lots of Gaussians: The EM algorithm

Fitting a single gaussian to the point set of *Fig.4.1* would leave out a lot of the structure. One way to improve the description of the point set would be to go to higher order moments, since we are using only second order moments here. If we do this we certainly get more information about the distribution of the points. Another approach is the *localise the data* approach. This amounts to fitting several ellipses, or if you prefer, several gaussians.

If you think of how you might describe handprinted characters as vectors, either by moments or some other means, then it is reasonable that much of the time two exemplars of the same character will be fairly close in the representation space. On the other hand, there are equally clearly cases where they won't; if for example half of the digits of a postcode are written by Europeans with a predilection for putting a horizontal bar across the vertical stroke of a seven, and the other half are written by Americans or British who have no such urge, then the sevens will form two quite distinct clusters in the representation space. Each cluster might, if we are optimistic, be representable by a single gaussian, but the two clusters together might not. So it would be as well to have several gaussians available for modelling digits, several for each digit according to the various strategies which are available for forming them.

The use of a *mixture of gaussians* is well recognised as a statistical technique, and the so called *EM algorithm* can be used to fit a fixed number of gaussians to a data set such as *Fig.4.1*. The bibliography contains pointers to books telling you the theory of handling mixture models quite generally. I shall concentrate exclusively on mixtures of gaussians, although it is not hard to see how to adapt the methods to other *pdf*s. It is found experimentally that the EM algorithm, although very fashionable, is very vulnerable to initialisation, and converges at a rate somewhat slower than a slug bungee-jumping in treacle. I shall have a little to say about methds of doing something about this a little later.

Given a data set such as that of *Fig.4.1*, it is not at all unreasonable to decide to fit six gaussians. It is also reasonable to wonder what one ought to do in higher dimensions where it might not be so easy to decide what a good number of gaussians would be. This is an important point I shall come to later, but for the time being let us suppose that we have a data set X of points all of the same category in $\mathbb{R}^n$, some number, *k*, of gaussians for distributing over the points, and we seek the maximum likelihood model. Algebraically, we have to find the maximum likelihood model for

$$\sum_{i=1}^{k} w_i g_{[\mathbf{m}_i, \mathbf{v}_i]}(\mathbf{x})$$

where the w_i are weights which have to be chosen so that the integral over the whole space is 1. So we have *k* weights, each a real number, *k* centres, each a vector in $\mathbb{R}^n$, and *k* symmetric positive definite

matrices, each *n* by *n*, each to be found so as to give the product of the likelihoods for each data point a maximum, that is:

$$\prod_{\mathbf{x}\in\mathcal{X}} \sum_{i=1}^{k} w_i g_{[\mathbf{m}_i,\mathbf{V}_i]}(\mathbf{x})$$

where of course each gaussian $g_{[\mathbf{m}_i,\mathbf{V}_i]}$ is a function of the form:

$$g_{[\mathbf{m}_i,\mathbf{V}_i]}(\mathbf{x}) = \frac{1}{(\sqrt{2\pi})^n \left(\sqrt{det(\mathbf{V}_i)}\right)} e^{-\frac{(\mathbf{x}-\mathbf{m}_i)^T \mathbf{V}_i^{-1} (\mathbf{x}-\mathbf{m}_i)}{2}}$$

and the whole has to be maximised by suitable choice of the weights and parameters for the gaussians. This is a rather horrid problem in constrained optimisation. Rather than try to solve it exactly, it is common to use the *EM* algorithm to find a solution. EM stands for `Expectation Maximisation' and describes the two steps in the algorithm. See the references for the origins and generality of the EM algorithm; here I shall give a procedure for obtaining the Maximum Likelihood Gaussian mixture model. I am most grateful to Arthur Nadas of the Automatic Speech Recognition group, IBM's Thomas J. Watson Research Center, Yorktown Heights for explaining this to me, among other things, in the dim and distant past.

- The EM algorithm for Gaussian Mixture Modelling

Next Up Previous Contents

Next: The EM algorithm for **Up:** Computing PDFs: Gaussians **Previous:** Dimension 2 *Mike Alder 9/19/1997*

The EM algorithm for Gaussian Mixture Modelling

1. Initialise the *k* gaussians so that each of the $\mathbf{V}_i$ is the identity matrix, each of the w_i is $\frac{1}{k}$, and the *k* centres are taken to be some small random step in $\mathbb{R}^n$ away from a randomly selected point of the data set X.

2. For each point $\mathbf{x} \in \mathcal{X}$, and for each *i* between 1 and *k*, compute the weighted likelihood $w_i g_{[\mathbf{m}_i, \mathbf{V}_i]}(\mathbf{x})$ using the last estimate of the parameters $w_i, \mathbf{m}_i, \mathbf{V}_i$. For each point $\mathbf{x}$, let $\mathcal{L}_i(\mathbf{x})$ be the likelihood assigned to it by the i^{th} member of the mixture:

$$\mathcal{L}_i(\mathbf{x}) = w_i g_{[\mathbf{m}_i, \mathbf{V}_i]}(\mathbf{x})$$

and let S_x be the sum

$$S_x = \sum_{i=1}^{k} w_i g_{[\mathbf{m}_i, \mathbf{V}_i]}(\mathbf{x}) = \sum_{i=1}^{k} \mathcal{L}_i(\mathbf{x})$$

and let

$$P_i(\mathbf{x}) = \frac{\mathcal{L}_i(\mathbf{x})}{S_x}$$

We think of the point $\mathbf{x}$ as ready to be split up between the *k* gaussians according to the *k* fractions

$$\frac{w_i g_{[\mathbf{m}_i, \mathbf{V}_i]}(\mathbf{x})}{S_x}$$

In consequence, we have that $\sum_{\mathbf{x} \in \mathcal{X}} \sum_{i=1}^{k} P_i(\mathbf{x}) = |\mathcal{X}|$, the cardinality of $\mathcal{X}$.

3. Now re-estimate the w_i as

$$w_i' = \frac{\sum_{\mathbf{x} \in \mathcal{X}} P_i(\mathbf{x})}{|\mathcal{X}|}$$

and the centres $\mathbf{m}_i'$ as the weighted centroids

$$\mathbf{m}_i' = \frac{\sum_{\mathbf{x} \in \mathcal{X}} P_i(\mathbf{x})\mathbf{x}}{\sum_{\mathbf{x} \in \mathcal{X}} P_i(\mathbf{x})}$$

and finally, the i^{th} covariance matrix, $\mathbf{V}_i'$ is re-estimated as

$$\mathbf{V}_i' = \frac{1}{(\sum_{\mathbf{x} \in \mathcal{X}} P_i(\mathbf{x}) \quad)} \sum_{\mathbf{x} \in \mathcal{X}} P_i(\mathbf{x}) \left[(\mathbf{x} - \mathbf{m}_i')(\mathbf{x} - \mathbf{m}_i')^T \right]$$

4. Run through steps **2** and **3** until there is no significant change in any of the numbers being estimated, then stop.

It may be shown that in principle this converges to a maximum likelihood mixture model for the data set X with *k* gaussians in the mixture. In practice it is found that the initialisation is rather critical and one does not always get good results. It is easy to experiment with data such as *Fig.4.1* in two dimensions, where the data comes from a known number of gaussians, and to discover that very unsatisfactory `solutions' are found by the above process, as *Fig.4.3* shows.

Figure 4.3: A `solution' found by the EM algorithm with poor initialisation.

The problem of suitably initialising the EM algorithm will be discussed later, as will the issue of finding the `right' value of *k*, the number of gaussians. In the chapter on Quadratic Neural Nets I shall describe a wholly dynamical approach to the business of modelling point sets which has affinities with mixture modelling.

Inspecting the data by projection can help in suggesting a suitable number of gaussians to employ in a mixture model, and can also suggest reasonable initialisations of the centres.

It is worth noting that the estimates of the eigenvalues obtained from diagonalising the covariance matrix are biased in an interesting way: the true ellipsoid is likely to be shorter and fatter than the estimate obtained from a data set of limited size. In other words the larger eigenvalues are overestimated and the smaller ones underestimated. One can accomodate this in various ways which are left to the imagination and enterprise of the reader.

Next Up Previous Contents

Next: Other Possibilities **Up:** Lots of Gaussians: The **Previous:** Lots of Gaussians: The *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Bayesian Decision **Up:** Computing PDFs: Gaussians **Previous:** The EM algorithm for

Other Possibilities

We might find cases where gaussians in arbitrary numbers look silly, perhaps by eyeballing the data under projection, perhaps by knowing something about the process which generated the data points, and it is a straightforward matter to select other families of distributions from those in the standard books introducing probability theory to the young. It is possible to choose some quantisation level and produce a histogram. And it is possible to use nearest neighbour methods as mentioned in chapter one.

It is also possible to prefer other than maximum likelihood models; for example we may want to use gaussian mixtures but have strong prejudices about where they should be, or to believe that the covariance matrices should all be equal, or something equally implausible *a priori*. It is not hard to make the appropriate changes to the algorithms to deal with all of the above.

In the next section, we suppose that we have obtained, by hook or by crook, some *pdf* for each of the categories of data point in the vicinity of a new data point, and we set about determining the category of the new point, that is we solve the standard supervised learning pattern classification problem.

Next Up Previous Contents

Next: Bayesian Decision **Up:** Computing PDFs: Gaussians **Previous:** The EM algorithm for *Mike Alder* *9/19/1997*

Bayesian Decision

Suppose now that we have, for each category of data point, modelled the set of points in that category by some suitable *pdf*, with a gaussian or a mixture of gaussians being likely to be suitable in practical cases quite a lot of the time. We have, in the vernacular, *trained* the model on the data, and now wish to use it on some new data, either to test the model to see how reliable it is or because we want to make an informed guess at what the category of a new datum is and this is our preferred method. I shall explain the ideas behind the best known decision methods.

Recall from the last chapter when we explained Bayes' Theorem and its use in decision, that we considered the case where $g_m(x)$ and $g_f(x)$ were the likelihoods produced by two competing models, one gaussian g_m for the guys and another g_f for the gals. We decided, you may recall, to regard these two numbers as the conditional probability of having got x given model m and the conditional probability of having got x given model f, respectively. We wrote this, with some reservations, as $p(x|m)$ and $p(x|f)$ respectively. These were interpreted by the sanguine, and us, as measuring the probability that the datum x will be observed given that model m (respectively, f) is appropriate. Now what we want is $p(m|x)$ and $p(f|x)$ respectively, the probabilities that the models are true given the observation x. By applying Bayes Theorem in a spirit of untrammelled optimism, we deduced that

$$p(m|x) = \frac{p(x|m)p(m)}{p(x)}$$

with a similar expression for $p(f|x)$.

In the situation of the first problem at the end of chapter one; you will doubtless recall, yet again if you read the last chapter, the half naked ladies and possibly the trees from which they were to be distinguished. Now it can be, and was, argued that in the absence of any data from an actual image, we would rate the probability of the image being of a tree as 0.9 and of it being a naked lady as 0.1, on the grounds that these are the ratios of the numbers of images. Bayesians refer to these as the *prior probabilities* of the events. So in the above formulae we could put $p(m)$ and $p(f)$ in as numbers if we had some similar sort of information about the likelihoods of the two models. This leaves only $p(x)$ as a number which it is a little hard to assign. Happily, it occurs in both expressions, and if we look at the *likelihood ratio*, it cancels out. So we have:

$$\frac{p(m|x)}{p(f|x)} = \frac{p(x|m)p(m)}{p(x|f)p(f)}$$

and the right hand side, known as the *likelihood ratio* is computable. Well, sort of.

We concluded in the last chapter that if you are prepared to buy this, you have a justification for the rule of thumb of always choosing the bigger value, at least in the case where $p(m)$ and $p(f)$ are equal. In this case, $p(m|x)$ is proportional to $p(x|m)$ and $p(f|x)$ is proportional to $p(x|f)$ and with the same constant of proportionality, so choosing whichever model gives the bigger answer is the Bayes Optimal Solution. More generally than the crude rule of thumb, if it is ten times as likely to be m as f which is responsible for a datum in a state of ignorance of what the actual location of the datum is, then we can use this to obtain a bias of ten to one in favour of m by demanding that the ratio of the likelihoods be greater than ten to one before we opt for f as the more likely solution.

- Cost Functions
- Non-parametric Bayes Decisions
- Other Metrics

Next Up Previous Contents

Next: Cost Functions **Up:** Decisions: Statistical methods **Previous:** Other Possibilities *Mike Alder 9/19/1997*

Cost Functions

In the argument for choosing between models *m* and *f* using Bayes' theorem, we had a prior probability for *p*(*m*) and *p*(*f*) and we could calculate *p*(*x*|*m*) and *p*(*x*|*f*). There is an extension to the idea which tries to choose not just the most likely model, but the model which will be the least expensive. If we make a decision to opt for *m* and we turn out to be wrong, there is, on this model, some cost associated with the error. Likewise if one turns out to be wrong in a choice of *f*. Presumably, being right does not incur any costs. If it does, it might be prudent to refuse to make a decision at all.

Suppose *C*(*m*) is a cost associated with choosing *m* and being wrong, and *C*(*f*) is a similar cost of wrongly choosing *f* under the impression it is right when it isn't. Suppose we are about to make a decision for a particular *x*, assigning it to *m* or *f*.

If we choose *m* and we are right, the cost is zero, if we choose *m* and we are wrong, then it is really an *f*, and the fraction of times it is an *f* is *p*(*f*|*x*). So a strategy of always choosing *m* on observing *x* would have an expected loss of :

$$\mathcal{E}(m,f) = p(f|x)C(m)$$

Similarly, the strategy of always choosing *f* after having observed *x* would have a total expected cost of:

$$\mathcal{E}(f,m) = p(m|x)C(f)$$

In order to minimise the total cost, at each decision point we simply choose the smaller of these two numbers. Making the Bayesian substitution to obtain things we can more or less calculate, we get that the rule is to pick the smaller of

$$\frac{C(m)p(x|f)p(f)}{p(x)}, \quad \frac{C(f)p(x|m)p(m)}{p(x)}$$

i.e. We get the *Bayes Classifier Rule*:

Choose *m* iff $\frac{p(x|m)}{p(x|f)} > \frac{C(m)p(f)}{C(f)p(m)}$.

In the event of there being more than two categories, some minor modifications have to be made, which will be left as an exercise for the diligent student.

The above analysis is rather simple minded; for a more sophisticated discussion see the paper *On the*

Derivation of the Minimax Criterion in Binary Decision Theory by Pyati and Joseph, as well as the following note by Professor H.V. Poor in the IEEE Information Theory Society Newsletter, Vol. 43, No.3. September 1993.

Next Up Previous Contents

Next: Non-parametric Bayes Decisions **Up:** Bayesian Decision **Previous:** Bayesian Decision *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Other Metrics **Up:** Bayesian Decision **Previous:** Cost Functions

Non-parametric Bayes Decisions

If we assume that there is a model *m* generating the points of category X and another, *f*, generating the points of category O in , then the above recipe can be applied providing we have estimates of the relative likelihoods of a new datum for each model, and prior probabilities for the models.

Using a neighbourhood and counting the fraction of the existing data which is of category X can be defended as giving an estimate of the likelihood function for model *m* in the neighbourhood. Taking the total number of points in each category as an estimate of the prior probabilities, we get that the likelihood ratio is simply the ratio of the different counts of the two categories in the neighbourhood. If the ratio stayed approximately the same as the size of the neighbourhood got smaller until the numbers were too small to inspire confidence in the estimate, then one might be inclined to buy this.

Similarly, we could find the closest k points to our new point and then count the fraction which are of category X to estimate the local *pdf*. The same kind of reasoning applies.

Observe that both of these methods presuppose that the metric is known and not negotiable. This may not be a convincing assumption. It is necessary to take a critical view of fundamental assumptions throughout this section: all too often there is something nasty hiding under the algebra.

Next Up Previous Contents

Next: Other Metrics **Up:** Bayesian Decision **Previous:** Cost Functions *Mike Alder*
9/19/1997

Other Metrics

Suppose we have fitted a gaussian to some data but have some difficulty in believing that the distribution is truly gaussian. We may nevertheless feel that in fitting a quadratic form to the data we have done *something* to say a little about how the data is distributed. For example, if we had the data of *fig*. 1.4 and a gaussian for each category of point, we might believe that it told us something about the right metric to use. Calculating likelihoods requires some assumptions about how the density of the data falls off with distance, and we may feel uncommitted to the gaussian model, but we may nevertheless feel that the gaussian model is telling us something about the choice of units and the way in which distances ought to be measured.

Given a gaussian distribution containing the quadratic form

$$(\mathbf{x}-\mathbf{m})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{m})$$

we may use this to calculate a number for any $\mathbf{x}$. If we get the result 1, we have said that the result is the distance of $\mathbf{x}$ from $\mathbf{m}$ is one standard deviation, generalising the one dimensional sense. In general the square root of the result for any $\mathbf{x}$ is called the *Mahalanobis distance* of $\mathbf{x}$ from $\mathbf{m}$ relative to the form $\mathbf{V}^{-1}$. Alternatively, the Mahalanobis distance is just the distance measured in standard deviations.

A *Riemannian Metric* on the space $\mathbb{R}^n$ is specified by assigning a positive definite, symmetric, quadratic form to every point of the space. If you have a curve in the space, you can take a set of points along it, and compute the Mahalanobis distance of each point from its predecessor, then add them up. Doing this with more and more points gives, in the limit, the length of the curve in the given metric. To actually compute the distance between two points in the given metric, take all curves joining them and find the length of the shortest.

The assignment of a quadratic form to each point of a space constitutes an example of a *tensor field*, and classical physics is full of them. General Relativity would be impossible without them.

We are not going to obtain a Riemannian metric from a finite data set without a good deal of interpolation, but there are occasions when this might be done. In the case of *fig*.1.4 for example, we can argue that there are grounds for putting on the same quadratic form everywhere, which is equivalent to squashing the space along the X-axis until the ellipses fitting the data turn into circles, and then using the ordinary Euclidean distance.

It is generally quicker to compute Mahalanobis distances (if not say them) than likelihoods since we save an exponentiation, and simply using the Mahalanobis distance to the various centres and choosing the smallest may be defended as (a) quick, (b) using a more defensible metric and (c) containing less

compromising assumptions. If the determinants of the different forms are all the same, this will give the same answers anyway. And if they do not differ by very much, we can argue that we are kidding ourselves if we are pretending they are really known accurately anyway, so why not go for the quick and dirty?

It is not uncommon to take the logarithms of the likelihoods, which amounts to just the Mahalanobis distance plus a correction term involving the determinant, and since the logarithm is a monotone function, this will give precisely the same answers as the likelihood if we are just choosing the greater likelihood source. The same simplification may be introduced in the case of gaussian mixture models.

Next Up Previous Contents

Next: How many things in **Up:** Bayesian Decision **Previous:** Non-parametric Bayes Decisions *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Overhead **Up:** Decisions: Statistical methods **Previous:** Other Metrics

How many things in the mix?

In the last chapter I sketched the theory of Rissanen on *Stochastic Complexity*, as described in his book Rissanen(1989) and papers (see the bibliography of the last chapter). As explained above, it offers a philosophically quite different approach to finding the `best' model to account for the data, and an approach which avoids some of the objections

I have pointed out to the classical theory. In particular it appears to solve elegantly the issue of how many gaussians one ought to have in a gaussian mixture model for a particular data set. (The answer may be `none', if the data set is small enough!)

As explained earlier, the general question as to order of a model occurs in other settings also. At the simple level of fitting a polynomial to a set of data points, the question as to what degree of polynomial is appropriate arises. At one extreme one can have the degree so high that every point is fitted, at the other we can assume it is constant. The same problem arises in the case of Neural Nets, as we shall see in a later chapter.

The gaussian mixture modelling problem, how many gaussians should we use, makes the problem quite stark. There will be an increase in the log-likelihood of the data with respect to the model whenever we increase the number of gaussians, until the degenerate case of non-invertible covariance matrices makes it infinite. But there is a penalty to be paid in terms of the number of parameters needed to specify the increasing number of gaussians. It is sensible to try to penalise the model in some way by offsetting the increase in log-likelihood with a corresponding increase in the number of parameters, and seeking to minimise the combination. But then the question arises, what should be the rate of exchange between log-likelihood and the parameter count? Rissanen gives us a natural and plausible rate of exchange. In this section I shall give some examples of computations on fitting gaussians in one dimension to data; see the discussion in the last chapter for the framework of thought which justifies the procedure.

- Overhead
- Example
- The Akaike Information Criterion
- Problems with EM

Next Up Previous Contents

Next: Overhead **Up:** Decisions: Statistical methods **Previous:** Other Metrics *Mike Alder*
9/19/1997

Overhead

Suppose we are agreed that we are going to use k gaussians in $\mathbb{R}^n$ for modelling our data, subject to the caveats given above on boundedness, we have no deep feelings about what k ought to be. If we are going to encode our data set relative to the *pdf* given by the k gaussians and their weights, we need to transmit or store not just the data encoded (N data points, each comprising n real numbers, each to precision using the gaussian mixture model but also (1) the number of gaussians k, (2) the k weights, real numbers encoded to some precision P', (3) the k centres, each vector in $\mathbb{R}^n$, and (4) the k covariance matrices, each having $n(n+1)/2$ values. We can make some plausible simplifications: let us assume the precision of the parameters, P', be the same as the precision of the components of the vectors, P. This merits some thought, let's duck the issue. Let us also suppose that the parameters can take any values inside some bounded region of $\mathbb{R}$ with a uniform *pdf*. let us also suppose that k is going to vary within some small range, from, say 1 to 20, uniformly. Thus we lose generality but make the sums easier for the kinds of cases we need to consider. Doing this in full generality is interesting statisticians, but we have more concrete problems in mind at present.

I shall suppose that the receiver knows what a gaussian is, and that I don't have to teach him what the idea of a function is, and so on. This is not a trivial matter if we believe that we learn to use some compression systems, but I simplify down to comparing between gaussian mixtures only. Extra overheads will therefore be the same and can be neglected in the calculation.

Then with these crude assumptions, I can code N data points in $\mathbb{R}^n$ directly by sending NnP bits. I assume here that the precision P affects all the components of the vectors, including the digits representing the integer part. If necessary, fill to the left with zeros. This is wasteful, but not as wasteful as sending you the length each time if the length variation is small.

If I send the mixture model, I get a fixed number of bits to specify k, and then kP bits for the weights, nkP bits for the centres, and $kPn(n+1)/2$ bits for the covariance matrices.

The total number of bits I get to use is therefore approximately

$$NPn + N\int_B f(x)\log_2\left(\frac{1}{f(x)}\right)dx + kP\left(\frac{n^2+3n+2}{2}\right)$$

where $f(x)$ is the gaussian mixture model discussed earlier, and where an additive constant for specifying k, which I shall suppose does not depend on k (because it will be the same for all the values of k within our range) has been omitted. I also take it that I am integrating the gaussian f over some bounded region B in $\mathbb{R}^n$. Naturally, I should take B to be centred on the region where the data is. The obvious choice for

B is to make it some fixed number of `standard deviations' from the *k* centres. A simpler choice is to take some hypercube within which the data may be found and use that. This is what I shall do.

If I have faith in my (truncated) model, then the amount of compression can be computed directly from the above formula: it represents the best compression for which I can hope, for data sets size *N* generated according to the model. A more cautious approach is to take the actual data merely try to compress that, using the model as a convenient device for doing so.

What we do here is to sum the logarithms of the improbabilities of each data point according the model. If each real number is specified to *P* bits, and if we have normalised everything inside the unit hypercube of dimension *n*, then there are 2^{nP} cells each having measure 2^{-nP}. The probabilities are the likelihoods multiplied by this last number. So the compressed data will have size in bits given by the sum

$$NPn - \sum_{\mathbf{x}\in\mathcal{X}} \log_2(f(\mathbf{x})) + kP\left(\frac{n^2+3n+2}{2}\right)$$

It must be remembered that the function *f* needs to be adjusted to have integral 1 over the hypercube into which the data has been compressed.

Now given two such functions, all we need to do to get maximum compression in the case where is the same and the data set fixed, is to make the sum of the log likelihoods as big as possible.

This of course maximises the likelihood product. If the two *f*s also differ in respect of the number *k*, the penalty for increasing *k* has to be balanced against an increase in the log likelihood sum. This gives you, according to Rissanen, the better of the two models.

Next Up Previous Contents

Next: Example **Up:** How many things in **Previous:** How many things in *Mike Alder*
9/19/1997

Example

Suppose we have data consisting of the 18 points:

$$\{-1.2, -1.1, -1.1, -1.0, -1.0, -1.0, -0.9, -0.9,$$

$$-0.8, 0.8, 0.9, 0.9, 1.0, 1.0, 1.0, 1.1, 1.1, 1.2\}$$

I decide to model this with two gaussians or maybe just one, on the interval from -2 to 2. I decide that I only care about the data accuracy to one part in 64, which is to say 6 bits. Rather than normalise, I shall assume that all data must come in the interval from -2 to 2. My first model for maximum likelihood with 2 gaussians will have equal weights and a centre at -1 with a variance of .015 and hence one standard deviation is 0.1225. The same applies to the second gaussian except that here the centre is at +1. With this value for σ (to 6 bit precision) the model for *f* is

$$\frac{1}{2}\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x+1)^2}{2\sigma^2}} + \frac{1}{2}\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-1)^2}{2\sigma^2}}$$

The likelihoods may be approximated (by ignoring the contribution from the other gaussians) as:

data point	likelihood	log-likelihood
-1.2	0.4293	-0.845567
-1.1	1.16699	0.154433
-1.1	1.16699	0.154433
-1.0	1.628675	0.487767
-1.0	1.628675	0.487767
-1.0	1.628675	0.487767
-0.9	1.16699	0.154433
-0.9	1.16699	0.154433
-0.8	0.4293	-0.845567
0.8	0.4293	-0.845567
0.9	1.16699	0.154433
0.9	1.16699	0.154433
1.0	1.628675	0.487767
1.0	1.628675	0.487767
1.0	1.628675	0.487767
1.1	1.16699	0.154433

```
1.1             1.16699          0.154433
1.2             0.4293          -0.845567
```

This gives the sum of the log-likelihoods as 0.78 which tells us that the model is pretty bad. I have taken natural logarithms here, if I convert to bits instead of nats, I get 1.1 bits.

The bit cost is therefore a total of

$$(18)(6) -1.1 + (2)(6)(3)$$

bits, that is 142.9.

The corresponding calculation for one gaussian has $\sigma \approx 1.06$ and all the likelihoods are around 0.22, so the sum of the log likelihoods is about -27.25 nats. This works out at around a 39.3 bit disadvantage. This should tell you that a bad model is worse than the uniform model, which is to say worse than sending the data. The bit cost of using a single gaussian centred on the origin therefore works out at

$$(18)\ (6) + 39.3 + (1)(6)(3)$$

that is, 165.3 bits. This is bigger, so it is better to use two gaussians than one to model the data, but it is even better to just send the data at a cost of 108 bits. You should be prepared to believe that more data that fitted the model more convincingly would give a genuine saving. Even *I* think I could believe that, although, of course, I can't be sure I could.

Better check my arithmetic, it has been known to go wrong in the past.

Next Up Previous Contents

Next: The Akaike Information Criterion **Up:** How many things in **Previous:** Overhead *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Problems with EM **Up:** How many things in **Previous:** Example

The Akaike Information Criterion

The Akaike Information Criterion (AIC) is another attempt to tackle the problem of working out how much to penalise the parameters of the model. It proceeds by making some rather strong assumptions about the family of models, and also that the data is in fact generated by some member of the family, which is a hierarchically ordered set of manifolds, each manifold embedded at height zero in its successor. By investigating the way in which the maximum likelihood model will overestimate the expected log-likelihood (by fitting itself to noise, just like you seeing patterns in the fire), Akaike came out with the rule:

(AIC) Maximise the function (Log-Likelihood(Data) - Number of parameters)

This is easy to compute for gaussian mixture models, and omits all reference to the question of the precision of the data or the parameters. All we have to do is to use EM to get the maximum likelihood model (but see below) and then compute the Log-Likelihood and subtract the number of parameters. Do this for a reasonable range of the number of gaussians and pick the maximum.

It will be seen that the only differences between the two formulae are the base of the logarithms (natural logs for the AIC) and the precision of the parameters, since the precision, dimension and number of the data are constant. Generally the Stochastic Complexity criterion will penalise the number of parameters more heavily than the AIC. The AIC is only appropriate when there is a fairly large data set.

In addition to the AIC, the BIC due to Scwartz and Bayesian arguments may also be used to compute penalties for having too many parameters, but it is a matter of opinion as to which of these, if any, is appropriate for any particular problem. See the paper by Speed and Yu, in the Bibliography for the last chapter, for a contemporary examination of the issues for a special case.

I shall return to the matter of compression and optimal modelling when we come to Neural Net models, which are notorious for having enormous dimensions for representing the data and consequently large numbers of parameters.

Next Up Previous Contents

Next: Problems with EM **Up:** How many things in **Previous:** Example *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Summary of Chapter **Up:** How many things in **Previous:** The Akaike Information Criterion

Problems with EM

As was remarked earlier, it is found experimentally that the EM algorithm, although very fashionable, is very vulnerable to initialisation, and converges at a rate somewhat slower than a slug bungee-jumping in treacle. Given the effective limits of precision of computers, this can simply lead to wrong answers, as was shown earlier.

Reasons for preferring EM are that it does, given good initialisation, lead to a Maximum Likelihood solution in many cases. Thus one can feel at peace with statisticians and probabilists, and assuage one's conscience about the optimality of the classification. To those who have read the last chapter, this has about as much force as chanting mantras and using garlic to keep vampires at bay, but the unreflective need all the confidence and certainty about life they can get. In some cases, the constrained optimisation problem which is what finding the Maximum Likelihood solution amounts to for a mixture model, can be solved by traditional methods, usually involving some numerical work and Newton's Method. I shall not go into this approach because it is found that with reasonable initialisation, EM works reasonably well, and I shall have something to say about Neural Model methods of accomplishing this a little later.

Next Up Previous Contents

Next: Summary of Chapter **Up:** How many things in **Previous:** The Akaike Information Criterion *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Exercises **Up:** Decisions: Statistical methods **Previous:** Problems with EM

Summary of Chapter

The chapter started with a review of the preceding chapters and a promise to discuss the standard statistical methods for classifying points. The first section however comprised a short homily on the merits of investigating the data by inspecting the numbers. Most of us who have worked with data collected by other people have cautionary tales of its unreliability and the general fecklessness of other folk. Let us make sure nobody speaks so of *us*. Let us, in fact, scrutinise all data in as many ways as possible, and approach it on the hypothesis that it has been collected by politicians. Let us in particular project the data down from dimension umpteen onto the computer screen, and inspect it as if it might be suffering from a social disease.

The next section was where I came good on the much more practical matter of how you did the sums. I showed how you could compute the ML Gausian for a set of data points in $\mathbb{R}^n$. If the data set doesn't look as if one category of data point is usefully describable by a single gaussian, I indicated that the *EM* algorithm can be used, after suitable initialisation, to get the ML gaussian mixture model. The idea is to take each cluster of points in one category and model the cluster as closely as possible by some number of gaussians.

The following section showed what to do when two clusters had been modelled by competing *pdf*s. It derived the Bayes Optimal Classifier rule. It also threw in, for good measure, a quick and dirty rationalisation for the use of neighbourhood counting methods, a discussion on the Bayesian modelling of data by choosing a prior *pdf* and having the data convert us to a posterior *pdf*.

Finally, I described how the Rissanen Stochastic complexity/ Minimum Description Length approach allows one to deal with the vexing question of the order of a model. I took some 18 points judiciously arranged in the interval [-2,2] and showed how by considering the overhead of specifying the model, it was possible to justify using two gaussians instead of one in a formal manner. I described the alternative AIC rather briefly but gave you a rule of thumb which can be applied. And *really* finally, I wrote this summary, the exercises and the bibliography.

Next Up Previous Contents

Next: Exercises **Up:** Decisions: Statistical methods **Previous:** Problems with EM *Mike Alder 9/19/1997*

Exercises

1. Write a program to generate some random dots in a more or less gausian distribution on the real line. Now modify it so that it generates a two dimensional approximately gaussian cluster so that if I specify the mean and covariance matrix I get a suitable point set. There are good and efficient ways of doing this, but yours can be quick and dirty. Next compute the mean and covariance of the data so generated. Does it bear a passing resemblance to what you started out with? Can you account for any discrepancies? Draw the one standard deviation ellipse for your data.
2. Use the above program to generate a `gaussian hand' as in Fig. 4.1. Try to use the EM algorithm with random initialisation to find a maximum likelihood fit for six gaussians. What conclusions do you draw about the algorithm?
3. Get someone to use your program to generate a mixture of gaussians and then try to fit the mixture without knowing how many terms in the mixture. Use the AIC and the Rissanen Stochastic Complexity measures to find the optimum number of model elements. Try this with varying numbers of points and varying numbers of mixture terms in various dimensions. Do the results surprise you? Depress you?
4. Photocopy an image of a real hand by putting your palm on the photocopier, cut it out, paint it black and scan it into a tif file. Use the EM algorithm to represent the pixel array as a gaussian mixture model. Can you find an initialisation process which gives reasonable results regardless of the orientation or position of the hand in the image? Are the results worse than for the gaussian hand?
5. The six quadratic forms specifying the components of a hand comprise six points in $\mathbb{R}^5$.Compute the centre and covariance matrix for these points. This gives 5 numbers for the centre and a further 15 numbers for the covariance matrix. This makes a hand into a point in $\mathbb{R}^{20}$. Repeat with a new handprint in the same location and the same orientation, and with the same handedness, i.e. stick to right hand prints, giving a set of points in $\mathbb{R}^{20}$. Now fit a gaussian to these points. Repeat by turning the paper hand over so as to reverse its `handedness'. Obtain a similar cluster for these mirror hands. Now test to see if you can classify left hands and distinguish them from right hands by looking to see which cluster they fall closest to. How badly do things go wrong if the fingers are sometimes together? (With judicious and ingenious initialisation, it is possible to get consistent

fits and reliable discrimination.)

6.

Repeat with paper cut-outs of aeroplane silhouettes, and see if you can distinguish between a fighter and a passenger jet. Make them about the same size, so it isn't too easy.

7.

Try gaussian mixture modelling an image of some Chinese characters. Can you use the same trick of describing the gaussians by computing gaussians for them (UpWriting) in order to tell different characters apart?

8.

Compute Zernike moments for some characters of printed English text and collect examples of the same character. Examine the clustering for variants of the characters by projecting down onto the computer screen (the programs with the book will save you some coding if you have a workstation). Compare the character `9' with a comma. Amend the Zernike moments to give a size measure.

9.

Construct a program which does printed Optical Character Recognition. You will find problems with characters joined together and characters which are disconnected; you will find limited font independence a problem. This is a lot of work but *very* educational, and later exercises will allow you to extend the scope of it considerably. It should be possible to get moderately good recogition results for clean text correctly oriented by several of the methods outlined in the preceding chapters.

Next Up Previous Contents

Next: Bibliography **Up:** Decisions: Statistical methods **Previous:** Summary of Chapter *Mike Alder 9/19/1997*

Bibliography

1. W. Feller, An Introduction to Probability Theory and its Applications Vols 1,2, New York: Wiley 1957, 1971.
2. D.M. Titterington, A.F.M. Smith, U.E. Makov , *Statistical analysis of finite mixture distributions* New York : Wiley, 1985.
3. B.S. Everitt and D.J. Hand, *Finite mixture distributions* New York: Chapman and Hall, 1981.
4. A.P. Dempster, N.M. Laird and D.B. Rubin, *Maximum Likelihood from Incomplete Data via the EM Algorithm*,Proc.Roy.Stat.Soc., 1, 1977.
5. Torfinn Taxt, Nils Lid Hjort and Line Eikvil, *Statistical Classification using a linear mixture of two multinormal Probability Densities* Pattern Recognition Letters 12 (1991) pp731-737.
6. James O. Berger, *Statistical decision theory, foundations, concepts, and methods* New York : Springer-Verlag, 1980.
7. James O. Berger, *Statistical decision theory and Bayesian analysis* 2nd ed. New York : Springer-Verlag, c1985
8. N.N. Cencov, (Tr: Lev J. Leifman), *Statistical decision rules and optimal inference* Providence, R.I : American Mathematical Society, 1982.
9. Enders A. Robinson, *Statistical reasoning and decision making* Houston, Tex : Goose Pond Press, 1981.
10. L. Weiss, *Statistical decision theory* New York : McGraw-Hill, 1961.
11. G. Hadley, *Introduction to probability and statistical decision theory* San Francisco : Holden-Day, 1967.
12.

Abraham Wald, *Statistical decision functions* New York : Wiley, 1950.

13. Anatol Rapoport, *Decision theory and decision behaviour : normative and descriptive approaches* Dordrecht, the Netherlands ; Boston : Kluwer Academic Publishers, 1989.

14. Geoffrey J McLachlan *Discriminant analysis and statistical Pattern Recognition* Wiley 1992.

15. Geoffrey J McLachlan and Kaye E. Basford, *Mixture models : inference and applications to clustering* M. Dekker 1988.

16. Bertrand R. Munier(Ed.), *Risk, decision, and rationality* Kluwer Academic Publishers, 1988.

17. Bruce W. Morgan, *An introduction to Bayesian statistical decision processes* Prentice-Hall, 1968.

Next Up Previous Contents

Next: Decisions: Neural Nets(Old Style) **Up:** Decisions: Statistical methods **Previous:** Exercises *Mike Alder*
9/19/1997

Decisions: Neural Nets(Old Style)

In the second chapter I suggested ways of turning an image or a part of an image, into a vector of numbers. In the last chapter, I showed, *inter alia*, how to model a collection of points in $\mathbb{R}^n$ by gaussians and mixtures of gaussians. If you have two or more categories of point (paint them different colours) in $\mathbb{R}^n$, and if you fit a gaussian or mixture of gaussians to each category, you can use the decision process (also described in the last chapter) to decide, for any new point, to which category it probably belongs.

It should be clear that in modelling the set of points belonging to one category by gaussians (or indeed any other family of distributions) we are making some assumptions about the nature of the process responsible for producing the data. The assumptions implicit in the gaussian mixture *pdf* are very modest and amount to supposing only that a large number of small and independent factors are producing unpredictable fluctuations about each of a small number of `ideal' or `template' stereotypes described by the measuring process. This is, frequently, not unreasonable: if we are reading printed text, we can suppose that there are several ideal shapes for a letter /A/, depending on whether it is italic, in Roman or sans-serif font, and that in addition there are small wobbles at the pixel level caused by quantisation and perhaps the noise of scanning. There should be as many gaussians for each letter as there are distinct stereotypes, and each gaussian should describe the perturbations from this ideal. So the approach has some attractions. Moreover, it may be shown that any *pdf* may be approximated arbitrarily closely by a mixture of gaussians, so even if the production process is more complex than the simple model suggested for characters, it is still possible to feel that the model is defensible.

If we take two clusters of points, each described by a gaussian, there is, for any choice of costs, a decision hypersurface separating the two regions of $\mathbb{R}^n$ containing the data, as in the figure. This hypersurface is defined by a linear combination of quadratics and hence is itself the zero of some quadratic function. In the particular case when both clusters have the same covariance matrix, this reduces to a hyperplane. If the covariance matrices are not very different, then a hyperplane between the two regions will still be a fair approximation in the region between the two clusters, which is usually the region we care about. And you can do the sums faster with an affine hyperplane, so why not use hyperplanes to implement decision boundaries? Also, we don't usually have any good grounds for believing the clusters are gaussian anyway, and unless there's a whole lot of data, our estimates of the covariance matrices and centres are probably shaky, so the resulting decision boundary is fairly tentative, and approximating it with a hyperplane is quick and easy. And for more complicated regions, why not use piecewise affine decision boundaries? Add to this the proposition that neurons in the brain implement affine subspaces as decision boundaries, and the romance of neural nets is born.

In this chapter, I shall first outline the history and some of the romance of neural nets. I shall explain the Perceptron convergence algorithm, which tells you how to train a single neuron, and explain how it was

modified to deal with networks of more than one neuron, back in the dawn of neural net research. Then I shall discuss the hiatus in Neural Net research caused by, or at least attributed to, Marvin Minsky, and the rebirth of Neural Nets. I shall explain layered nets and the Back-Propagation algorithm. I shall discuss the menagerie of other Articial Neural Nets (ANNs for short) and indicate where they fit into Pattern Recognition methods, and finally I shall make some comparisons with statistical methods of doing pattern Recognition.

There was an intereresting exchange on the net in 1990 which concerned the relative merits of statistical and neural net (NN) models. Part of it went as follows:

`... I think NNs are more accessible because the mathematics is so straightforward, and the methods work pretty well even if you don't know what you're doing (as opposed to many statistical techniques that require some expertise to use correctly)' .....

However, it seems just about no one has really attempted a one-to-one sort of comparison using traditional pattern recognition benchmarks. Just about everything I hear and read is anecdotal. Would it be fair to say that ``neural nets'' are more accessible, simply because there is such a plethora of `sexy' user-friendly packages for sale? Or is back-prop (for example) truly a more flexible and widely-applicable algorithm than other statistical methods with uglier-sounding names? If not, it seems to me that most connectionists should be having a bit of a mid-life crisis about now.'

From Usenet News, comp.ai.neural-net, August 1990.

This may be a trifle naive, but it has a refreshing honesty and clarity. The use of packages by people who don't know what they're doing is somewhat worrying, if you don't know what you're doing, you probably shouldn't be doing it, but Statistics has had to put up with Social Scientists (so called) doing frightful things with SPSS and other statistical packages for a long time now. And the request for some convincing arguments in favour of Neural Nets is entirely reasonable and to be commended. The lack of straight and definitive answers quite properly concerned the enquirer.

The question of whether neural nets are the answer to the question of how brains work, the best known way of doing Artificial Intelligence or just the current fad to be exploited by the cynical as a new form of intellectual snake oil, merits serious investigation. The writers tend to be partisan and the evidence confusing. We shall investigate the need for a mid-life crisis in this chapter.

- History: the good old days
  - The Dawn of Neural Nets
  - The death of Neural Nets
  - The Rebirth of Neural Nets
  - The End of History
- Training the Perceptron
  - The Perceptron Training Rule
- Committees

- Committees and XOR
- Training Committees
- Capacities of Committees: generalised XOR
- Four Layer Nets
- Building up functions

- Smooth thresholding functions
  - Back-Propagation
  - Mysteries of Functional Analysis
  - Committees vs Back-Propagation
- Compression: is the model worth the computation?
- Other types of (Classical) net
  - General Issues
  - The Kohonen Net
  - Probabilistic Neural Nets
  - Hopfield Networks
    - Introduction
    - Network Characteristics
    - Network Operation
    - The Network Equations
    - Theory of the Network
    - Applications
  - The Boltzmann Machine
    - Introduction
    - Simulated Annealing
    - Network Characteristics
    - Network Operation
    - Theory of the Network
    - Applications
  - Bidirectional Associative Memory
    - Introduction
    - Network Characteristics
    - Network Operation
    - The Network Equations

        - Theory of the Network
        - Applications
    - ART
        - Introduction
        - Network Characteristics
        - Network Operation
        - Theory of the Network
        - Applications
    - Neocognitron
        - Introduction
        - Network Structure
        - The Network Equations
        - Training the Network
        - Applications
    - References
    - Quadratic Neural Nets: issues
- Summary of Chapter Five
- Exercises
- Bibliography

Next Up Previous Contents

Next: History: the good old **Up:** An Introduction to Pattern **Previous:** Bibliography *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: The Dawn of Neural **Up:** Decisions: Neural Nets(Old Style) **Previous:** Decisions: Neural Nets(Old Style)

History: the good old days

- The Dawn of Neural Nets
- The death of Neural Nets
- The Rebirth of Neural Nets
- The End of History

Mike Alder
9/19/1997

Next Up Previous Contents

Next: The death of Neural **Up:** History: the good old **Previous:** History: the good old

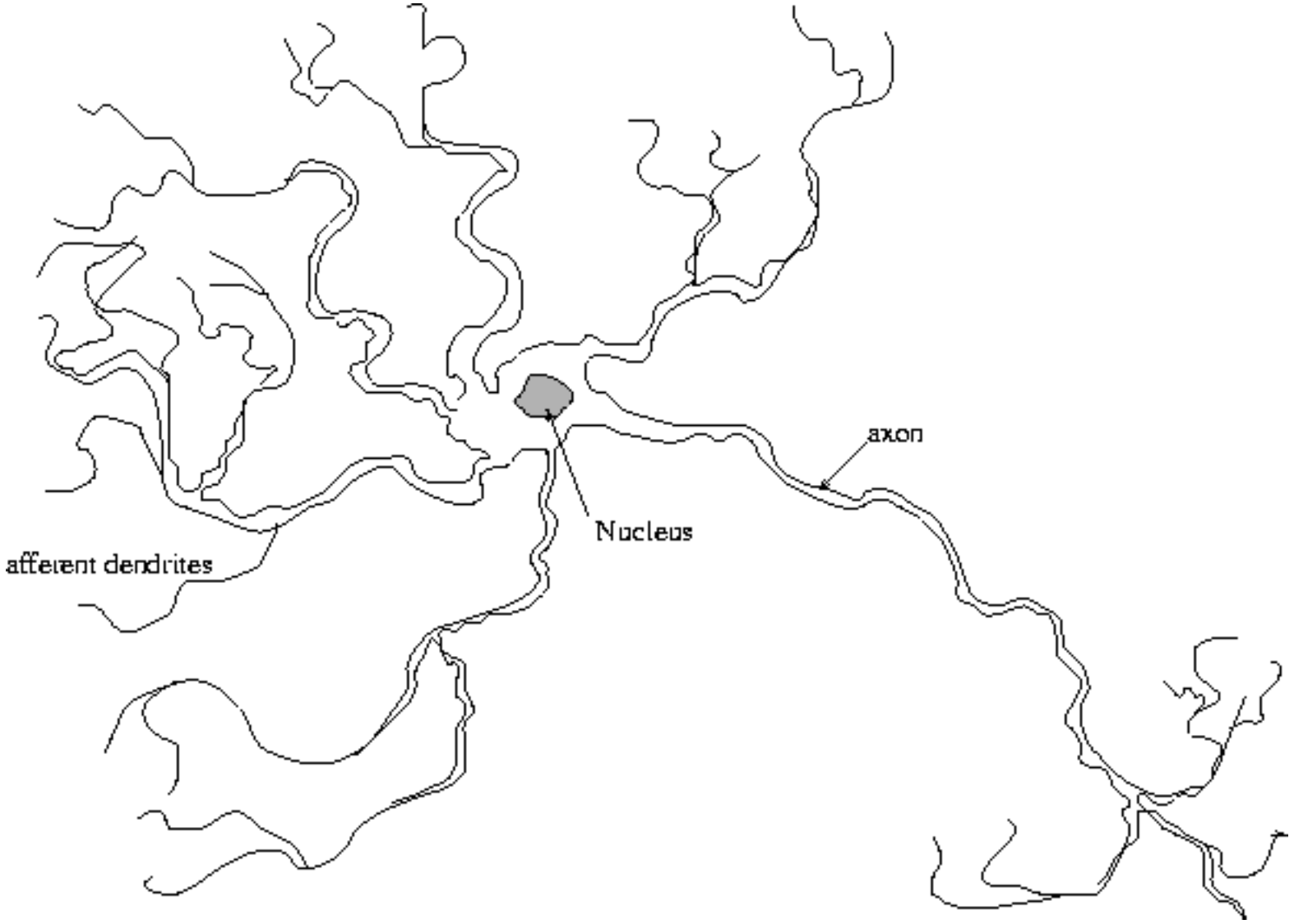
The Dawn of Neural Nets

In the `classical' theory of Neural Nets, which we may trace to Rosenblatt, Block, Widrow and Nilsson, not forgetting the early work of Pitts and McCulloch, there are two strands. One is the idea of piecewise affine (or piecewise linear as they used to say in the bad old days) hypersurfaces as discriminating boundaries, and the other is the belief that this is how brains might do the job of classifying data. I have already discussed the former in the case of the archetype pattern recognition problem of telling the guys from the gals.

The idea that a neuron might be a *threshold logic unit*, that is to say a unit where some weighted sum of inputs was compared with a threshold and either output a 1 or -1, according to whether the weighted sum exceeded the threshold or not, goes back to the observation that sometimes neurons fired, in which case a spike of voltage difference between inside and outside the neuron travelled along it, from input end to output, afferent dendrites to efferent dendrites, until it came to another neuron, at which point the following neuron, given a sufficient amount of stimulus from the first and possibly other neurons, might also fire. So the firing of a neuron came to be seen as an all or none affair, mediated by the transmission coefficients at the *synapses*, the places where one neuron impinged on another. This made it a *weighted majority gate*, in an alternative terminology, with each weight representing some state of a synapse. The *Hebb doctrine*, that memory was stored in transmitter substance density and possibly neuronal geometry, also contributed to the view of neurons as adaptive threshold logic units. This is not a view which has survived, but it had a substantial effect on the thrust of the development of algorithms in the classical period from about 1940 to the sixties.

The sketch of a `typical' neuron, as in *Fig.5.1* is a biological version of a big brother of *Fig.1.6*, with more inputs. Engineers tend to prefer straight lines and right angles to the wiggly curves of biology.

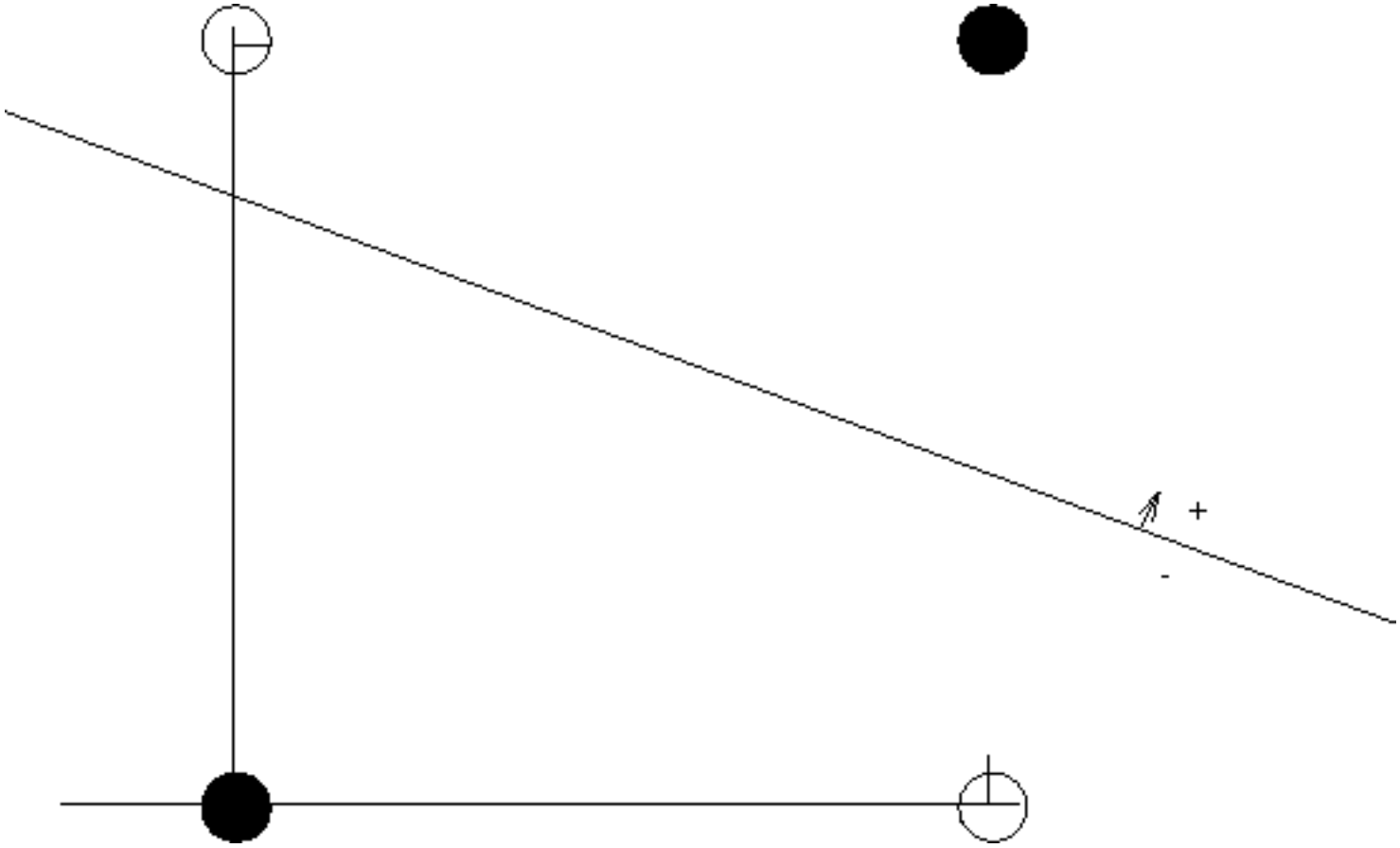
Figure 5.1: A sketch of a real, live neuron. Well, sort of.



Rosenblatt, in 1961, had seen the *elementary perceptron*, just a single one of these units, as a model for a single neuron, and conceded without shame that the critical issue of how more units could combine to give larger perceptrons was still open. He used a particular adaptation rule called the

alpha rule for training his elementary perceptron to output either +1 or -1, for use as a Pattern Classifier. It was well known that if the inputs were taken from the so called *XOR* data set, which had points at (0,0), (0,1), (1,0) and (1,1) where the first and last were of category 0 and the middle two of category 1 (and the name being chosen for reasons obvious to anyone with any familiarity with logic) then the elementary alpha perceptron could not classify the data correctly. This is rather simple geometry. The elementary perceptron with input the coordinates of points in the plane was simply an oriented line which fired if the input data was on one side and didn't if it was on the other. And the XOR data set was not linearly separable.

Figure 5.2: XOR: a classification problem beyond the capacity of the single neuron alpha-perceptron.



Nilsson in 1965 had shown how what he called a `committee net' of three neurons, all fed the same input, could all feed output to a final neuron, the final neuron seeing the preceding layer as voting on the correct classification of the datum. By simply taking the majority verdict, easily accomplished by a suitable choice of weights, the final neuron could yield the correct classification. We shall return to this shortly.

Next Up Previous Contents

Next: The death of Neural **Up:** History: the good old **Previous:** History: the good old *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: The Rebirth of Neural **Up:** History: the good old **Previous:** The Dawn of Neural

The death of Neural Nets

The late sixties saw the publication of Minsky and Papert's *Perceptrons*. This casually identified all perceptrons in the sense of Rosenblatt with the elementary alpha-perceptron, preceded by some local process which `extracted features' from an image by looking at only some of the pixels within a region, and returned a vector of binary results which was then the input to a single model neuron. This did less than justice to the possibilities for (a) looking at the whole image and (b) using many units and hence having piecewise affine functions instead of just affine functions. By pointing out the limitations of this rather skeletal abstraction in a, perhaps, rather confusing manner, Minsky and Papert managed to convince most readers that there were easy problems which their brains could solve which the perceptron model family could not. The cover of the book showed two diagrams, one a double spiral and the other a single spiral, and explained that it was hard for a perceptron to tell the difference. It was pretty hard for the reader, too, and involved some tracing of paths with fingers, but this was overlooked. Similarly, it was shown that combining local viewpoints could never guarantee to tell if a pixel set is connected or not. Rather like the proof that no (finite state) computer can add any two numbers, which it somewhat resembles, this is a result of limited practical utility. The whole range of all possible perceptrons, in the sense of Rosenblatt, would encompass what are now called recurrent nets as well as time delay nets and indeed just about all the extant neural nets. Minsky clobbered only a limited class, but he used Mathematics, and so hardly anybody troubled to read the fine print. What he was in fact trying to tackle was the issue of when local information about the geometry of an object could be condensed by a linear or affine map and then combined with other such local compressions to give global properties of the object. There is some reason to doubt that this is a good problem. There is even more reason to doubt if this problem has a lot to do with how brains work. Still, it is no dafter than much Mathematical research and was harmless in all respects except its effect on Neural Net work.

The moral to be drawn from this would seem to be that most people don't read books very carefully, especially if they contain mathematics. It might have helped if Minsky hadn't had such a high reputation, or if he had taken more care to explain that the limitations of the systems of the sort he was analysing were not pertinent to the entire model class. Minsky took it that the input to a perceptron was a discrete retina always, and that the values were binary only, and that it was impractical to look at other than local regions or to use other than affine functions. Thus the limitations of the perceptrons Minsky analysed were fairly severe from the beginning, and it was unfortunate that sloppy reading left the many with a vague notion that *all* perceptrons were being disposed of. There is a suspicion that the critical tone of the book was the result of a disenchanted love affair: the AI community had imagined that all their problems would be solved by just bunging them into a huge neural net and waiting for the computer to grind to a conclusion. Ah well, the sixties were like that. If MIT had Perceptrons, the hippies had flower power. Both sobered up eventually. MIT AI Lab did it first, but at the expense, perhaps, of a bad case of the snits.

The belief that Minsky had shown that Perceptrons couldn't do anything non-trivial, came at about the

same time that disenchantment was beginning to set in among funding officers. It is folk-lore that Rosenblatt had spent around Two Million US Dollars by the time of Minsky's book and had little to show for it, at least by the standards of the military. So it is possible that Minsky merely drove the nail a little further into the coffin. Anyway, *something* around this time more or less stopped research on piecewise affine systems dead in its tracks. But you can't keep a good idea down, and eventually by changing

the name to Artificial Neural Nets (ANNs) and later MultiLayer Perceptrons (MLPs) (when the variety of these fauna started to proliferate), they came back. They are now the staple of snake oil merchants everyhere, alas, and we need another Marvin Minsky to come again and tell us what they can't do. But this time, do it right.

For Minsky's comments on the dispute, see *Whence Cybernetics*, **Connections**, The Newsletter of the IEEE Neural Networks Council, Vol.3. No.3., p3., September 1993. And also P4. It should be remarked that some of the views expressed (by the original disputants, not by Minsky) show an extensive innocence.

Next Up Previous Contents

Next: The Rebirth of Neural **Up:** History: the good old **Previous:** The Dawn of Neural *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: The End of History **Up:** History: the good old **Previous:** The death of Neural

The Rebirth of Neural Nets

The belief of Minsky in the late seventies appeared to be that what was needed was an effective way of training multilayer perceptrons. The quick fix committee nets of Nilsson were ignored for reasons that seem to be accidental. Training a single neuron by the method of Rosenblatt (and Widrow, and others) didn't appear to generalise in a way that people thought right and natural. Certainly, training the committee net was found experimentally not to always work, but the *ad hoc* fix of adding more units usually managed to give success. Just why was not entirely clear. And then came the the idea of following the linear summation by a smooth approximation to the step function that had been used up to this point. This allowed one to use a little elementary calculus, together with a few minor fudges, to train nets of arbitrary complexity. Actually, what it really did was to give a rationale for doing what was being done anyway, but such is the power of myth that a belief was born that the new neural nets were the answer to the prayers that the Artificial Intelligentsia muttered to their heathen and doubtless robotic gods. The triumph of hope over experience seems to be an enduring feature of human existence.

The *Back-Propagation* algorithm then allows one to train multilayer perceptrons, the species of Neural Net to which you were introduced in the first chapter. They don't always converge to a solution, but it is found experimentally that adding a few extra units often works, just as with committee nets. This is not a coincidence. It is also found that the net can learn only rather simple patterns in a reasonable time, and that huge times are needed for the case where the data set is at all complicated. I shall discuss the algorithm a little later, and explain why it behaves as it does.

At the same time, the neurophysiologists were finding out what was being done and were less than enthusiastic about MLPs as models of anything inside anybody's head. The evidence against the core idea of neurons as adaptive weighted majority gates (which fired provided that the weighted sum of inputs exceeded some threshold), has been accumulating. While some may possibly work in this way, there seems to be much more than that going on in brains.

Other types of neural net were investigated around this time, and a number have survived. I shall deal with them rather briefly in later sections. Among them, the Kohonen net, the Adaptive Resonance net of Grossberg, the Probabilistic Neural Nets of Specht, the Hopfield nets and various other `architectures' such as the *Neocognitron* are going to get some attention.

In surveying the plethora of Neural Nets at the present time, there is a strong sense of having entered a menagerie, and moreover one on which no taxonomist has been consulted. I shall try to reduce the sense of anomie and focus on what kinds of things these nets accomplish, rather than on drawing wiring diagrams, which appear to be the main conceptual tool offered to the perplexed student.

My collaborator of many years, Dr. Chris deSilva, has written an elegant account of certain of these and has kindly given me permission to include a substantial article he has written explaining how some of them work.

Next Up Previous Contents

Next: The End of History **Up:** History: the good old **Previous:** The death of Neural *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Training the Perceptron **Up:** History: the good old **Previous:** The Rebirth of Neural

The End of History

The above history constitutes a somewhat brief and stark introduction to the subject of ANNs or Neural Networks; it is, however, lacking in that corroborative detail required to add artistic verisimillitude to an otherwise bald and unconvincing narrative. The remainder of the chapter will explain to those who can use the packages, just what they are doing when they use them.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: The Perceptron Training Rule **Up:** Decisions: Neural Nets(Old Style) **Previous:** The End of History

Training the Perceptron

It is simplest to return to the two dimensional case of Fred and Gladys from chapter one in order to find out what is happening with the single unit alpha-perceptron. I hear alarm bells ringing; they warn me that doing the solution in the case of dimension two with only two categories, and indeed only two data points, will convince the reader he is not getting his money's worth. That there has to be more profundity than this, or why do all the other books have so much more algebra? To conceal the lack of insight of the author, dear reader. Trust me.

Recall *Fig. 1.7*, which I have recycled as *Fig.5.3* and the more or less obligatory network diagram *Fig.1.6*, which I have recycled as *Fig.5.4*.

Figure 5.3: Two different states, A and B, of a neural net.

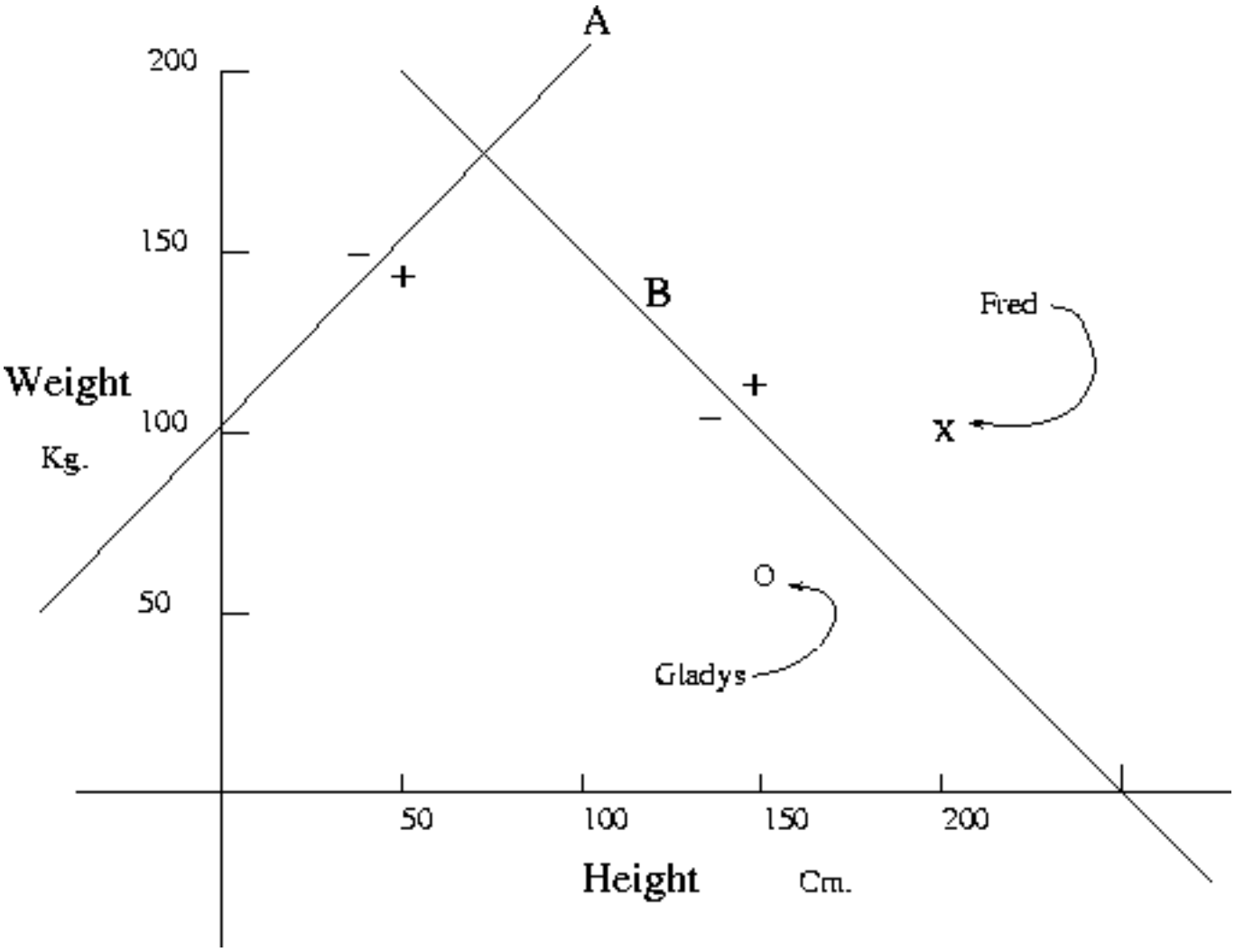
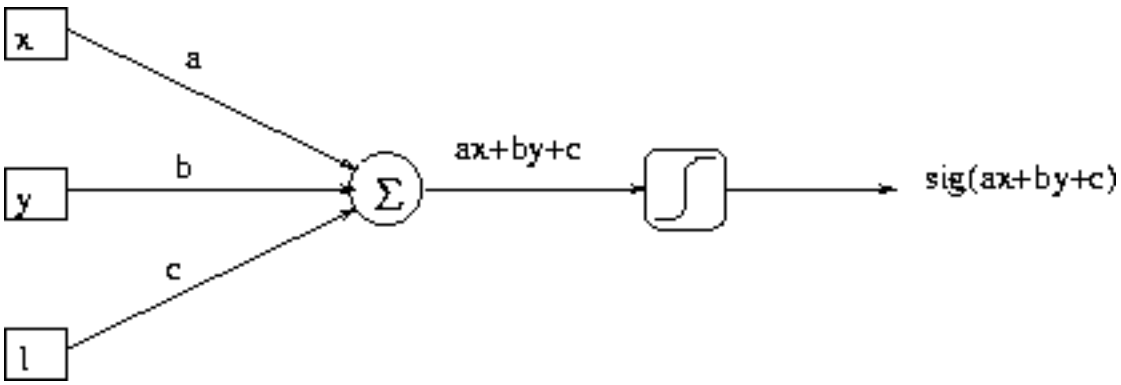


Figure 5.4: The neural net of the above diagram, in general.



Now let us initialise the line $ax + by + c = 0$, or if you prefer the numbers a,b,c at random inside some interval. I shall suppose that none of them are zero, as seems reasonable, and if you wish to choose to have each of the numbers between, say, -1 and +1, I should not mind at all, although I have no particular preferences.

If you imagine that line A of *Fig.5.3* is the result of this intialisation, then it will give us a concrete picture of the start of operations.

It is worth noting that in dimension *n*, I should have to choose *n*+1 numbers to specify a hyperplane. We have not yet given a name to the initial line or hyperplane, so I shall remedy this immediately by calling it $\mathbf{a}$. It is not a very romantic name for anything, not even a hyperplane, but it is at least short.

Next I select a point from the data set, also at random. In our application this means picking either Fred or Gladys. In general there would be, one might hope, some rather larger pool of talent. Since I don't wish to prejudice the discussion by leading the reader to believe that it matters which one I pick, I shall call my randomly selected data point $\mathbf{x}$. If you want to ask awkward questions about randomness, I evade them by requiring only that your pseudo-random number generator be reasonably uniform in its selections and not, for example, too flagrantly sexist in the case of Fred and Gladys.

Next I shall work out on which side of the line $\mathbf{a}$ the point $\mathbf{x}$ lies. The simplest algebraic way of specifying this is to *augment* the point $\mathbf{x}$ to be a point $\hat{\mathbf{x}}$ in $\mathbb{R}^{n+1}$ which has all the components of the vector $\mathbf{x}$ with a 1 inserted in the $(n+1)^{th}$ place. If we do this simple trick, then the summation of the inputs *x* and *y* weighted by *a* and *b* with *c* added on, is simply expressed by the dot product $\mathbf{a} \cdot \hat{\mathbf{x}}$. Now I threshold this value to take the sign of it, +1 if $\mathbf{a} \cdot \hat{\mathbf{x}} \geq 0$, -1 if $\mathbf{a} \cdot \hat{\mathbf{x}} < 0$. That is to say, I obtain $sgn(\mathbf{a} \cdot \hat{\mathbf{x}})$. The reader will note that although he is probably thinking of $\mathbf{a}$ as a line and $\hat{\mathbf{x}}$ as a point to which a 1 has been appended, there is no reference to the dimension in these expressions, so they make sense for points and hyperplanes in $\mathbb{R}^n$ for any *n* whatever.

Next I want to think about the category or class of the datum, $\mathbf{x}$.This is either +1 or -1 in my system of representing such things; It is a value which belongs with the point and so I shall define the map from the data set of points $\mathcal{D}$ to ± 1

$$c : \mathcal{D} \longrightarrow S^0$$

which assigns to each point $\mathbf{x}$ its class.

Note that this makes it fairly clear that the point of this business is to produce a map of the form $sgn(\mathbf{b} \cdot -)$ for some neural state $\mathbf{b}$, which takes the point $\mathbf{x}$ to the number $sgn(\mathbf{b} \cdot \hat{\mathbf{x}}), \pm 1$.

This map will be defined for any $\mathbf{x}$ in $\mathbb{R}^n$, so it will tell us the class, 1 or -1 of every point $\mathbf{x}$. It is of course essential that this map agrees with *c* on the data set and extends it to all other points in $\mathbb{R}^n$. In essence, this is not different in principle from my giving you a dozen x and y values and asking you to draw a smooth curve through them all; you are fitting a function to a set of data. Whereas you usually want to fit a smooth function, perhaps a polynomial, here you have to fit a function which takes values in

S^0, and which is given on points in $\mathbb{R}^n$, but you are still function fitting. That is, you have a finite function given you, and you want to extend it to the whole space. The curve fitting aspects may be emphasised by returning briefly to the diagram *fig*.1.11 of chapter one. Instead of fitting a smooth sigmoid as shown there, we would be fitting a step function, but the idea is the same. In dimension two, we should be fitting a step that runs right across the plane. Clearly we can focus on the dividing line, or more generally the separating hyperplane.

Now either $c(\mathbf{x}) = sgn(\mathbf{a} \cdot \hat{\mathbf{x}})$ or it doesn't. If it does, then we may say that the line $\mathbf{a}$ *correctly classifies* $\mathbf{x}$, or that $\mathbf{x}$ is on the right side of the hyperplane $\mathbf{a}$. Under these conditions we do absolutely nothing about changing the line $\mathbf{a}$.

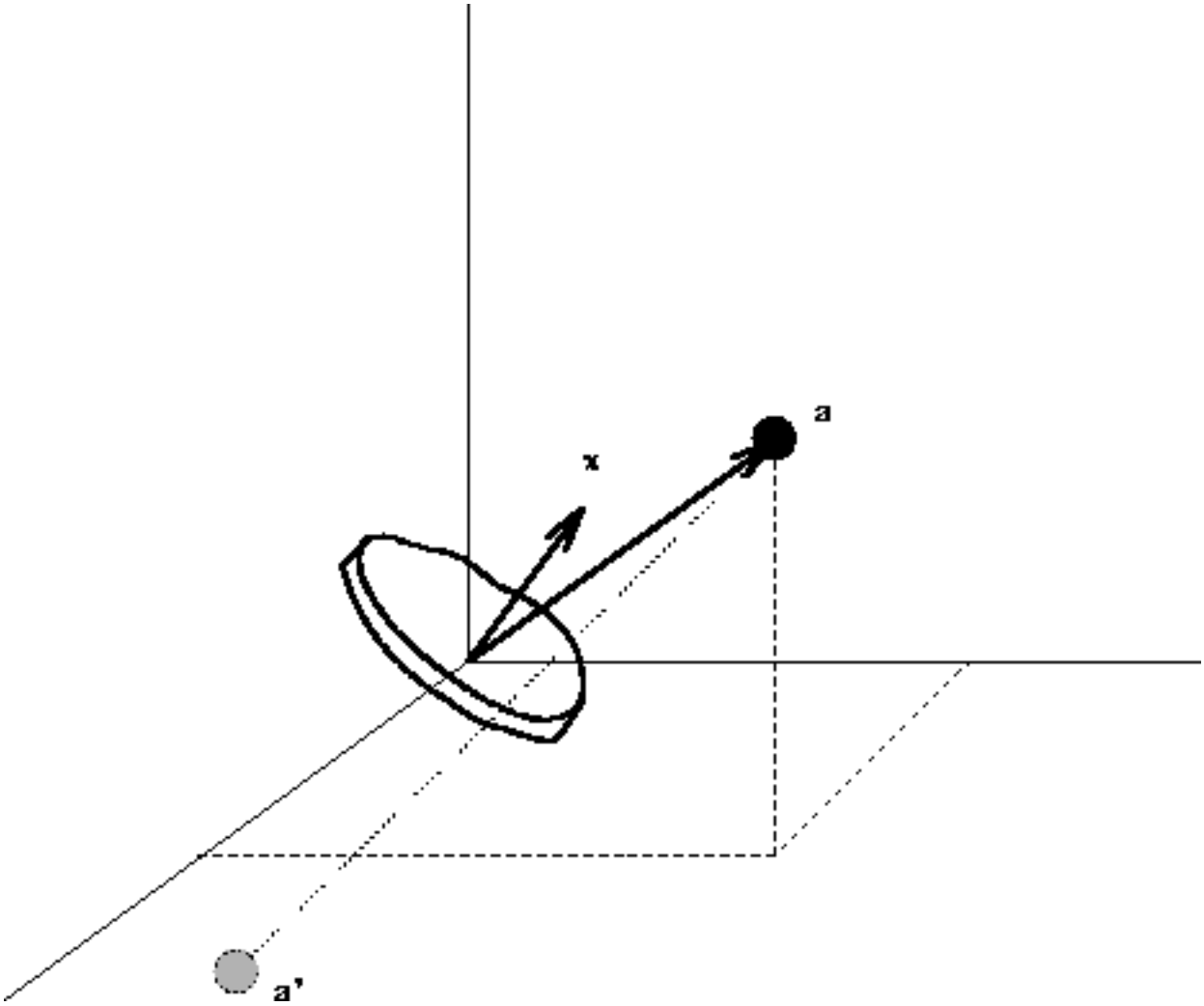
The other possibility is that $c(\hat{\mathbf{x}}) = -sgn(\mathbf{a} \cdot \hat{\mathbf{x}})$. In this case we need to beat $\mathbf{a}$ around a bit and change its state. It is a naughty neuron which is getting the wrong answer, so it should be smacked. None of this post modernist Spockery here, no praising the neuron for being right and talking gently to it when it is in error; if it is right we ignore it and if wrong we kick it.

To see how to kick it is an interesting exercise in data representation. The AI fraternity, who tell us that the representation makes a big difference, are right.

The idea is to observe that there are a lot of possible $\mathbf{a}$ lines, and there has to be a rule for operating on each such $\mathbf{a}$ and turning it into a more compliant, better behaved $\mathbf{a}$. So we look at the space of all such $\mathbf{a}$. In the case of the recalcitrant line, the space is the space of three numbers *a*,*b*,*c*. The actual line $\mathbf{a}$ is a point in this space. This may be confusing at first, but it is easy enought to see that $2x + 3y + 4 = 0$ can be represented in the space of all such lines as the vector $\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}$.

Now the line has been turned into a point in this space of all lines in the plane, and the datum $\hat{\mathbf{x}}$ also defines a point in this space. And the latter point gives the right dot product with half the points in the space, and the wrong dot product with the other half. And for the case when the dot product is zero, the turn around occurs. In this dual space of lines, otherwise called the *weight space*, $\{\mathbf{a} : \hat{\mathbf{x}} \cdot \mathbf{a} = 0\}$ is a plane through the origin. It divides the space of lines into two parts, those that are right and those that are wrong. So it is an error to think of the augmented data point as being a vector in the space of all lines. It is better to think of it as a plane through the origin in that space. I draw a rather unconvincing picture of a bit of it in *Fig. 5.5.*

Figure 5.5: Into the dual space.



Now given that the line $\mathbf{a}$ is on the wrong side of the plane through the origin normal to the datum point $\hat{\mathbf{x}}$ (and if you can take that without blinking you are either on top of this or not with us at all), the obvious thing to do is to reflect it through the plane until it is on the right side. And this is the Perceptron convergence algorithm. Moreover, this way of thinking about it, in the weight space, gives some insight into why it works, which beats memorising the recipe.

To do the sums is straightforward and requires only the most elementary linear algebra. Suppose $c(\mathbf{x}) = 1$. Since the neural state $\mathbf{a}$ is wrong about the point, we must have that $\hat{\mathbf{x}} \cdot \mathbf{a} < 0$. The projection of $\mathbf{a}$ on $\hat{\mathbf{x}}$ is pointing in the direction opposite to $\hat{\mathbf{x}}$ and is the vector $\frac{\mathbf{a}\cdot\hat{\mathbf{x}}}{\hat{\mathbf{x}}\cdot\hat{\mathbf{x}}}\hat{\mathbf{x}}$. We need to subtract twice this vector from $\mathbf{a}$ if $\mathbf{a}$ is to be reflected through the plane through the origin normal to $\hat{\mathbf{x}}$ and made into a better and purer line which is right about the data point $\mathbf{x}$.

If $c(\hat{\mathbf{x}}) = -1$, then $\hat{\mathbf{x}} \cdot \mathbf{a} > 0$ since the line is wrong about the point, and so the projection of $\mathbf{a}$

on $\hat{\mathbf{x}}$ is positive and shouldn't be. Both are on the same side of the normal plane through the origin. So again we subtract the vector

$$2\frac{\mathbf{a}\cdot\hat{\mathbf{x}}}{\hat{\mathbf{x}}\cdot\hat{\mathbf{x}}}\hat{\mathbf{x}}$$

from $\mathbf{a}$. The Perceptron Training rule is then:

- The Perceptron Training Rule

Next Up Previous Contents

Next: The Perceptron Training Rule **Up:** Decisions: Neural Nets(Old Style) **Previous:** The End of History *Mike Alder*
9/19/1997

The Perceptron Training Rule

From a random initial state of the unit, $\mathbf{a}$ and with a randomly selected data point, $\mathbf{x}$, of category given by $c(\mathbf{x})$, augmented to $\hat{\mathbf{x}}$ by appending a 1 in the n+1th place,

1. evaluate $\mathbf{a} \cdot \hat{\mathbf{x}}$. If $sgn(\mathbf{a} \cdot \hat{\mathbf{x}}) = c(\mathbf{x})$ select a new data point at random and repeat from the last value of $\mathbf{a}$.
2. If $sgn(\mathbf{a} \cdot \hat{\mathbf{x}}) \neq c(\mathbf{x})$, alter the state $\mathbf{a}$ to $\mathbf{a}'$ given by:

$$\mathbf{a}' = \mathbf{a} - 2\frac{\mathbf{a} \cdot \hat{\mathbf{x}}}{\hat{\mathbf{x}} \cdot \hat{\mathbf{x}}}\hat{\mathbf{x}}$$

   Rewrite $\mathbf{a}'$ to $\mathbf{a}$.
3. Choose a new data point $\mathbf{x}$ at random and go to 1.

It remains only to prove that if we keep on selecting points from the data set at random, and if the set of points is *linearly separable*, that is, if it is possible to find a line which correctly classifies the data set, then the iteration of the above process will eventually terminate with every data point correctly classified. This is the *Perceptron Convergence Theorem* and it is fairly easy to understand.

Having explained the ideas, we go through the process more formally.

Definition 13342

A *data set* is a finite subset $\mathcal{D}$ of $\mathbb{R}^n$, together with a map

$$c : \mathcal{D} \longrightarrow \mathcal{C}$$

into a finite set $\mathcal{C}$ of *categories*. If the set $\mathcal{C}$ has only two elements, the data set is said to be a *binary data set*, and $\mathcal{C}$ is written $\{1, -1\}$ and called S^0.

Definition 13348

A binary data set $\mathcal{D} \subset \mathbb{R}^n$ is said to be *linearly separable* iff there is a hyperplane in $\mathbb{R}^n$ such that all points of $\mathcal{D}$ of category +1 are on one side of the hyperplane, and all points of category -1 are on the other side.

Definition 13351

The space of all oriented hyperplanes in $\mathbb{R}^n$ specified by the n+1 weights on the arcs of a perceptron is called the *weight space*.

Theorem 13353 (Perceptron Convergence)

If $\mathcal{D} \subset \mathbb{R}^n$ is a binary data set which is linearly separable, then the Perceptron Training Rule will terminate in a finite number of moves with probability 1.

Proof. In the weight space, each augmented data point determines a hyperplane through the origin, and also a right side and a wrong side for this hyperplane, the right side consisting of the hyperplanes which correctly classify the point. The solution space of all hyperplanes which correctly classify all the data is simply the intersection of all these half spaces.

By hypothesis this is not empty, and except in a probability zero case, the intersection has strictly positive measure.

Take a ball on the origin in the weight space, and suppose, without loss of generality, that the initial state lies within this ball. Then the operation of making a change to the state of the perceptron is a folding of the ball about a hyperplane, and the inverse image by this folding of the solution space also has positive measure, and is a region of fixed positive measure removed from the set of states in the ball. Provided that all the data points operate in this way sufficiently often, the measure of the whole ball must be removed, whereupon every state in the ball is transformed to a solution state. □

It is easy to devise variants of the Perceptron training rule which are also guaranteed to give a solution from any initial state. For example, instead of the reflection, we could simply take a fixed step in the right direction, which is in the direction of $\hat{\mathbf{x}}$ if the category of $\mathbf{x}$ is +1 and the value of $sgn(\mathbf{a} \cdot \hat{\mathbf{x}})$ is -1, and is in the opposite direction when both signs are reversed. This will not correct the error in one move in general, but it will do so if iterated, and the idea of the proof still works: every correction takes a bite out of the space of states so that some that were wrong are now right and will never change again because they are right about all the data. And a finite number of bites of fixed size eats up any bounded set of initial states. This variant of the Perceptron training rule is known as the *delta* rule.

Note that we have the theorem in any dimension. Generalising it to more than two categories may be done in several ways, and is left as an exercise for the diligent reader.

Next Up Previous Contents

Next: Committees **Up:** Training the Perceptron **Previous:** Training the Perceptron *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Committees and XOR **Up:** Decisions: Neural Nets(Old Style) **Previous:** The Perceptron Training Rule

Committees

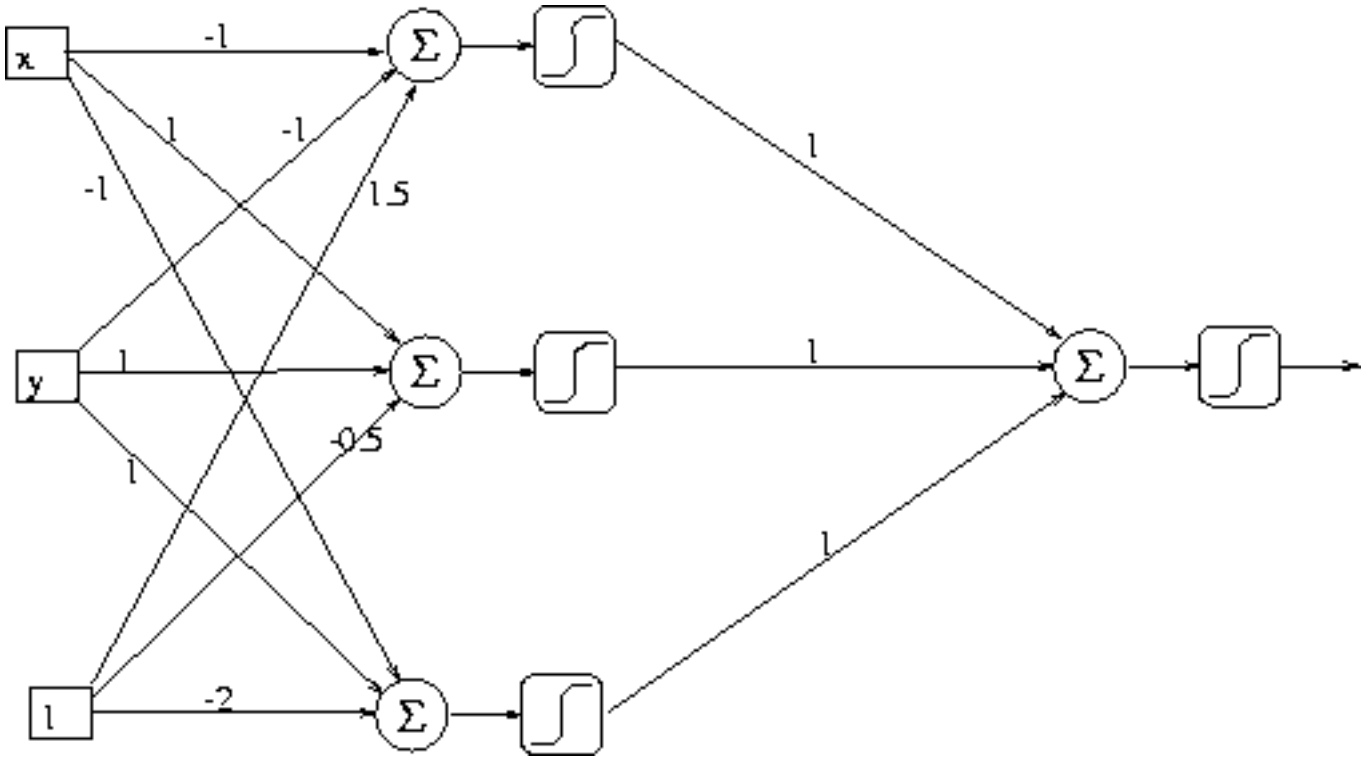
- Committees and XOR
- Training Committees
- Capacities of Committees: generalised XOR
- Four Layer Nets
- Building up functions

Mike Alder
9/19/1997

Committees and XOR

The XOR data set of *Fig.5.2* cannot be solved by a single unit because it is not linearly separable. It can however easily be solved by three units sharing input from points in the plane and one unit receiving its input from the first three. The diagram for the net is shown in *Fig.5.6.* It is called a committee, terminology due to Nilsson, because the last unit can be thought of as taking the votes of the three preceding units and adding them up. This will give the majority opinion since each vote is ± 1 .

Figure 5.6: A committee of model neurons.



This committee is referred to in the literature as a `three layer net; you might count only two layers, the square boxes being inputs not processing elements, but the traditions of the literature are against you. The fact that the literature appears to have been written by people who couldn't tell the difference between an input and a processing element may depress you somewhat, but look on the bright side. It can't really be a difficult subject.

Now we work through the points one at a time and trace the behaviour through the net. It is

recommended that you do this with a pen and paper to check my arithmetic, which is very shaky.

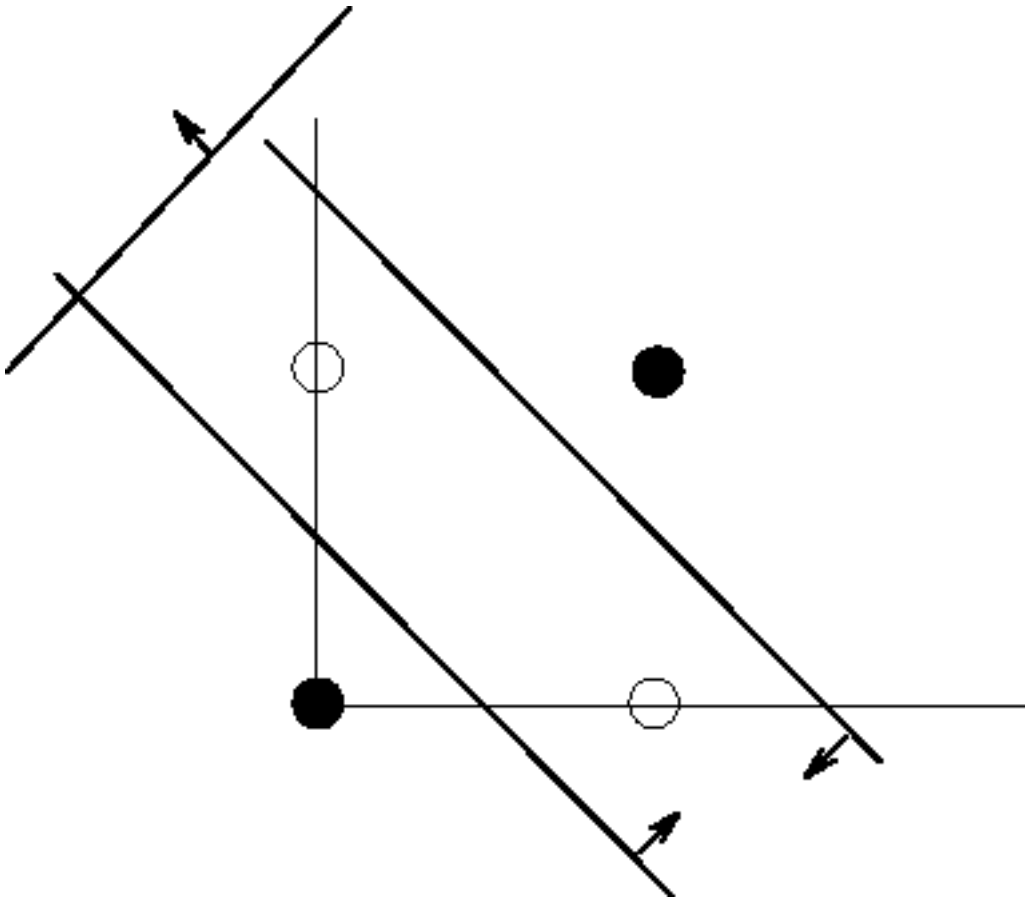
Suppose we input the point (0,0) to the net. The first unit outputs +1, the second and the third -1. The fourth unit in the third layer simply sums these and outputs the sign of the result which is -1. If we input the point (1,1) then the first unit outputs -1, the second +1 and the third -1 again, so the last unit again outputs -1.

If we input (1,0) then the first unit outputs +1, the second unit outputs +1 and the third unit outputs -1 again. So the majority vote sum of these numbers is +1. Similarly if we input (0,1), the first two units output +1, the third outputs -1 and the last unit outputs +1. So with the interpretation +1 means TRUE and -1 means FALSE, the network implements the logical function XOR.

So a committee of three units can do what a single unit cannot. The question which should spring to the mind of even the most intellectually numb is, how did I get the numbers above?

The answer is that I drew a simple diagram of lines and read off the coefficients of the lines. The picture I drew was that of *Fig.5.7*, and I simply surrounded the positive points with some positive lines. In fact the negative line is only there as a courtesy; it seemed easier to have an odd number of voters to ensure that there can't be any ties as a general principle.

Figure 5.7: A committee solving XOR.



Next Up Previous Contents

Next: Training Committees **Up:** Committees **Previous:** Committees *Mike Alder*
9/19/1997

Training Committees

If the net of *Fig.5.6* is in error about the XOR data set, how do we adapt it? Let us contemplate simple heuristic arguments first, as a prelude to understanding the Back-Propagation algorithm.

First, if the net is wrong about a point, we may suppose without loss of generality that the net thinks it a -1 when it is really a +1. So if the weights, call them *u*,*v*,*w*, on the last unit are all +1, then two or three of the lines have the point on the wrong side. We could therefore adapt the system by taking one of the lines that is wrong about the point, and simply use the Perceptron Training rule to change that line.

As an alternative, we could change the weights *u*,*v*,*w*. If we had all three lines wrong, we should have to change the sign of at least one weight, which is exactly equivalent to swapping the corresponding line's orientation, while leaving it in the same place. It seems unreasonable to keep the lines fixed and merely adjust their weights, since if the lines are simply in the wrong places, they will stay in the wrong places. In fact we have simply done a fixed transform to the input and then fed the result into a single unit. We might be lucky and have done by good luck a transform which improves things, but it would be silly to bet on it. So mucking about with the weights *u*,*v*,*w* is not a particularly attractive option. So we may reasonably prefer to keep the voting strengths the same and at +1, and concentrate on moving the lines.

This raises the question of which line to move. Now a line is either right or it's wrong, but it could be argued that it can be said to be very wrong in some cases, but not so wrong as all that in others. The conservative thing to do, is to make whichever change is smaller. The rationale for this is that presumably some of the data set has been learnt by the system, even if only by chance, and that it would be silly to throw away everything, so work out which change in the state space is least and do that one. This amounts to the following strategy:

take the lines $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ and the point $\hat{\mathbf{x}}$ such that $\mathbf{a}_i \cdot \hat{\mathbf{x}} < 0$ when the category $c(\mathbf{x}) = 1$,

and find the line $\mathbf{a}_i$ such that

$$\mathbf{a}_i \cdot \hat{\mathbf{x}} < 0 \ \& \ |\mathbf{a}_i \cdot \hat{\mathbf{x}}|$$

is a minimum. Then correct this $\mathbf{a}_i$ using the Perceptron training rule. Similarly for the case where

$c(\mathbf{x}) = -1$.

Experiments show that this works fairly well, and much better than such strategies as choosing the 'maximally wrong' line. A committe of three members such as *Fig.5.6* can find a solution fairly quickly in practice, for such trivial data sets as the XOR. It is easy to write a little program which draws the lines on

the screen and you can watch them hop about spasmodically until they converge to a solution. There is no advantage and some disadvantage in also modifying the weights u,v,w. Higher dimensional analogues to XOR can be constructed by taking a binary string of length n and assigning the category +1 to it if there is an even number of 1s in the string, and a -1 if there is an odd number. This `parity' data set has 2^n points and it is an amusing exercise to compute the smallest number of units in the second layer (committee members) required to ensure a solution.

Such committees, started off from random initial locations, can sometimes fail to converge, and a little innocent experimenting rapidly shows why. Suppose that two of the three wrong lines are `a long way away' from the datum $\hat{\mathbf{x}}$, and indeed from all the data. Then the same line may be corrected every time a datum is presented to the net, and the two remote units might just as well not be there. The correction process ignores them. In this case we are trying to solve XOR with a single unit, which is impossible. A `long way away' simply means that $|\mathbf{a}_i \cdot \hat{\mathbf{x}}|$ is large. This will happen for XOR if the constant term is big in absolute value.

There are several ways of fixing things up so that all units are employed usefully. One quite attractive procedure is the following: we do not simply select the closest unit for correction, we correct all of them, but we correct the closest unit most and the remoter units much less. Thus the most remote of units will eventually be pulled into the region where the data is being classified by the hyperplanes until the data is classified correctly, except where there are simply not enough units to do the job. Since we seldom know what this minimum number of units is except in rather trivial problems such as the parity problem, the prudent thing to do is to start with a lot and keep your fingers crossed.

The choice of correction rule which does this differential weighting of correction according to how remote the line is from the datum in the weight space, needs a function which starts off at some positive value and then decreases monotonically to zero. Something like e^{-x} springs immediately to mind (since distances are always positive!). It is not a particularly good choice, because it takes much longer to compute transcendental functions than polynomials or rational functions, and something like $1/(1+x)$ is easier to compute. This function has the property that remote objects get pulled in rather fast, which may be a bad idea if they are not so remote as all that, and are in fact doing a good job of classifying some other data points. One can therefore have a preference for, say, $1/(1+x)^2$. The possibilities are endless.

The end result of this purely heuristic analysis then, is that a workable correction process to the committee net can be obtained by keeping the final weights fixed at +1, and adapting the wrong units by taking a step in the appropriate direction (i.e. along the direction $\hat{\mathbf{x}}$ or in the opposite direction according to the sign of the category) of size some constant times a function which is decreasing to zero from 1 (or some other positive constant) and might conveniently be chosen to be e^{-x} or $1/(1+x)^2$. It is easy to explore this training rule on real data and it seems to behave itself sensibly. We shall see later that this rule is intimately related to the back-propagation procedure.

Next Up Previous Contents

Next: Capacities of Committees: generalised **Up:** Committees **Previous:** Committees and XOR *Mike Alder*

9/19/1997

Capacities of Committees: generalised XOR

It is intuitively clear, and easy to prove, that *some* committee can classify any finite data set in $\mathbb{R}^n$. I do this now, so you can brood over the proof.

Proposition 13407

Any finite binary data set in $\mathbb{R}^n$ can be classified by some committee net.

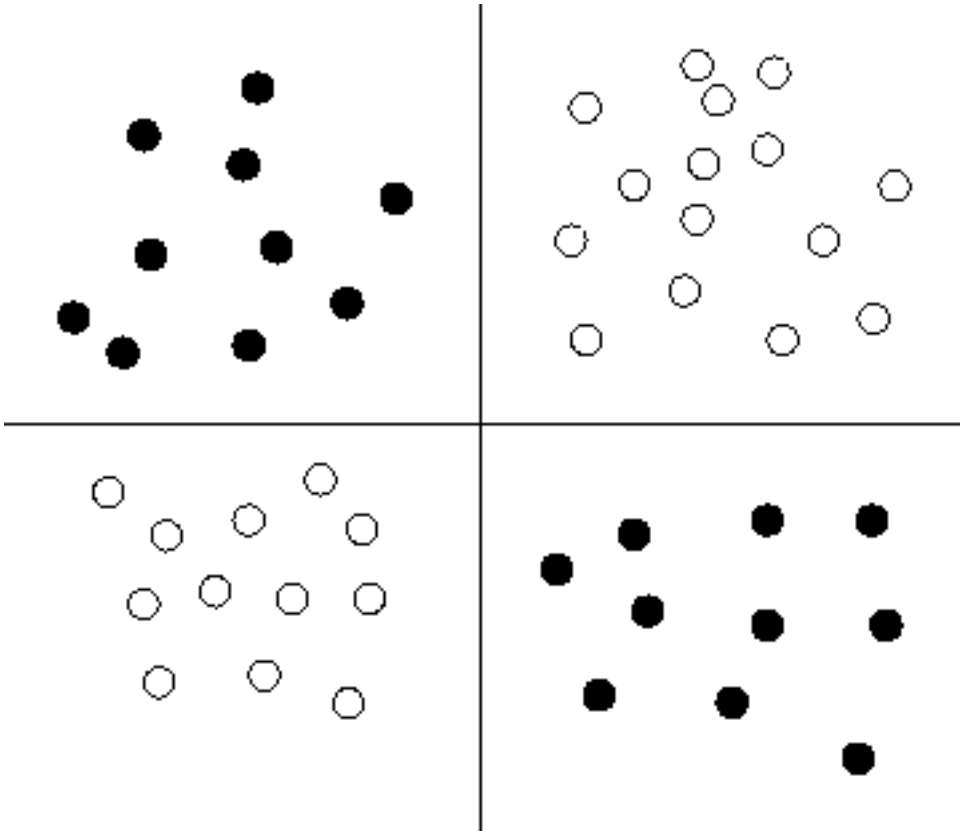
Proof We induct on the number of data points, N. It is clear that when $N = 1$ there is no problem, indeed it can be accomplished with one unit when $N \leq n+1$. Suppose then that all data sets having N data points in $\mathbb{R}^n$ an be classified by some committee, and consider a data set having N+1 points. One of these, call it p, has to be on the outside, that is must be separable from the remaining data set by a single hyperplane. There is a committee net which can classify correctly the remaining N data points, by the induction hypothesis. If the same net is correct in its classification of p there is nothing to prove, so we suppose that the committee assigns the wrong category to p.

Since p is separable from the rest of the data set, we may surround p by hyperplanes which do not intersect the convex hull of the remainder of the data set and which change the net's opinion of p, by putting them between p and the rest of the data, and we may put an equal number on the other side of p to ensure that outside this band containing p no change is made. The result is a committee which is correct about all the N+1 data points. The Principal of Mathematical Induction completes the proof. □

Note that the size of the committee may have to be increased faster than the size of the data set. This should, after some exposure to the ideas of Rissanen, make one sceptical about the extent to which such modelling actually accomplishes anything. I shall quantify this later.

The point is brought home if one considers a more `realistic' data set than XOR. Suppose one defines a *generalised XOR* data set in $\mathbb{R}^2$ by assuming a large number of points in the plane, all the points in the first and third quadrants have category -1 and all the points in the second and fourth quadrants have category +1.

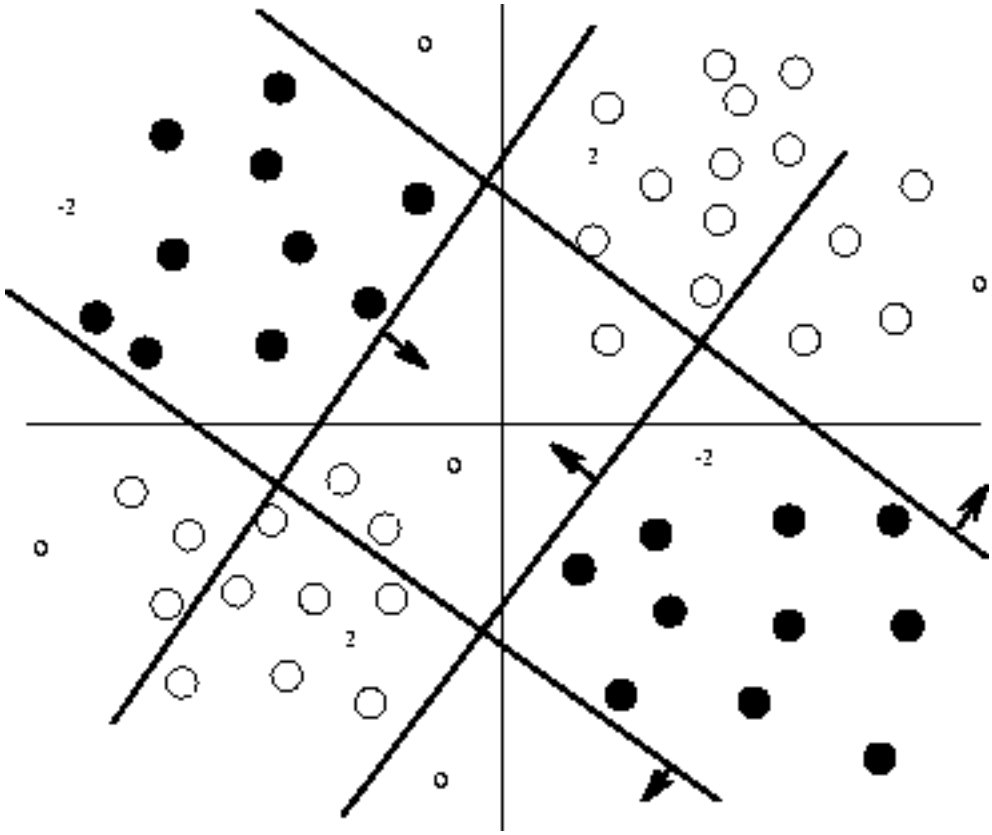
Figure 5.8: Generalised XOR.



An example of such a data set is sketched in *Fig. 5.8*, where the different categories are represented by circles black and circles white. Generalisation to higher dimensions is left as an easy exercise for the diligent student. Generalisation to more than two categories is also left as an exercise for the diligent reader.

Now the smallest number of oriented lines which can correctly classify the data set looks to be, in general, on the large side. A solution to the particular distribution of points of *Fig*. 5.8. is offered in the next sketch, *Fig.5.9*.

Figure 5.9: ....and a solution...



The numbers refer to the votes for and against. An additional unit or a suitably chosen threshold of 1 would make the discrimination.

Now it is easy to see that putting more data points in the four quadrants, according to the arrangement whereby points in the third and first categories are of type +1 and points in the second and fourth are of type +1, will soon start putting this model in trouble. The difficulty is quite plain: the units define lines across the whole plane, the rule for alternate quadrants has different regimes in half planes, and to continue briskly across the whole space is inevitably going to screw things up. Moreover, this is a generic problem not confined to the generalised XOR data set; if a data set consists of a collection of convex regions in $\mathbb{R}^n$, with each convex set containing points of the same category, and if the convex sets are chosen so as to make their number minimal, then if there are more than four of them, they will look like the generalised XOR set or worse.

The situation is evidently most unsatisfactory; it is easy to arive at the situation where one is trying to fit pairs of lines in so as to sneak through the regions largely occupied by the black circles, without actually enclosing any between the lines, with the intention of actually enclosing a single white circle. And the problem arises from the nature of the committee. Proposition 5.1 therefore is of limited practical value; the existence of generalised XOR and the fact that we can expect data sets which are structurally similar whenever the complexity of the pattern is high enough, tells us that three layer committees can be expected to perform badly as the number of units will be of order the number of data points. As we shall see later, the three layer MultiLayer Perceptron with a sigmoidal squashing function (and any algorithm for finding a solution) is subject to the same carping comments. The predictive power of such a system will be close to zilch. The thoughts on compression and information extraction lead us to doubt if the model implemented by such a net is worth having, a point to which I shall return later.

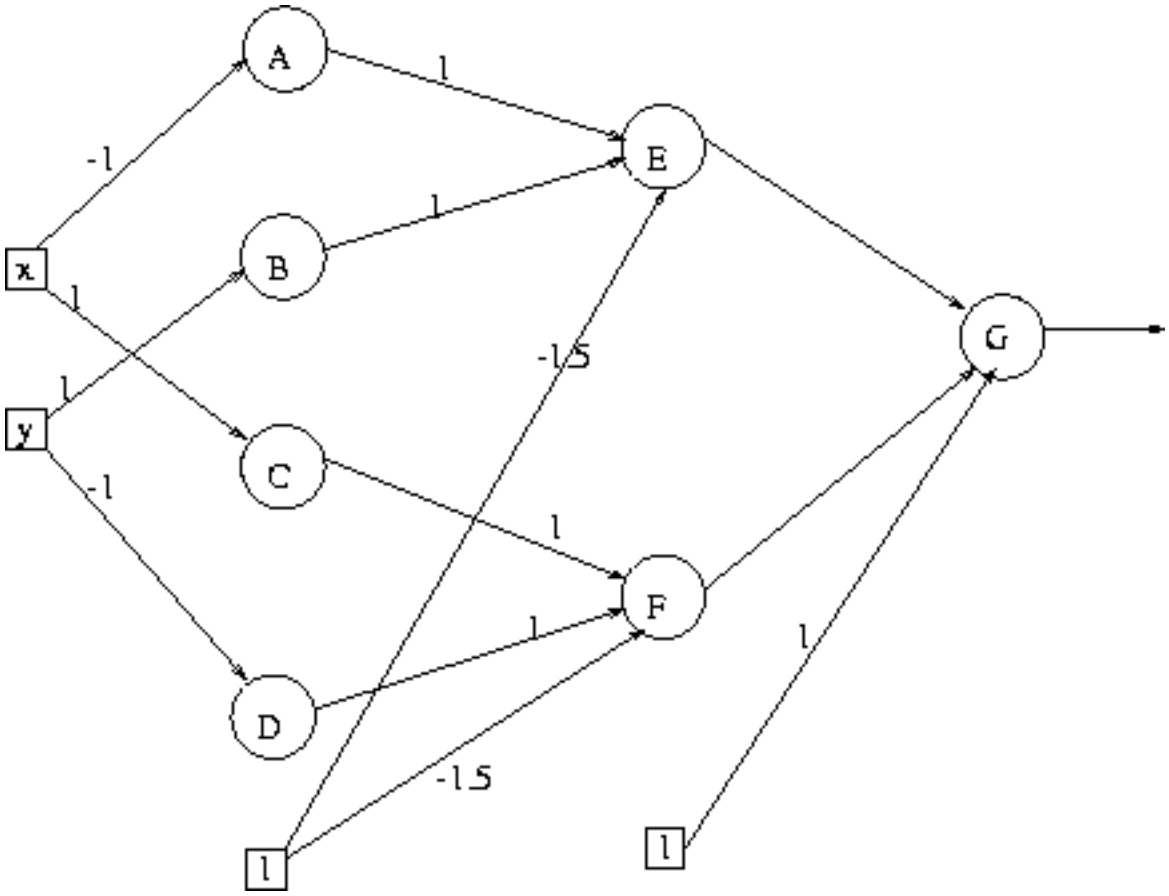
Next Up Previous Contents

Next: Four Layer Nets **Up:** Committees **Previous:** Training Committees *Mike Alder*
9/19/1997

Four Layer Nets

There is a simple solution to the problem of the last subsection. We simply add another layer. The next diagram, *Fig.5.10*, gives a diagram of a net which solves the generalised XOR problem of *Fig 5.8*. Each node is supposed to do a weighted linear summation of the inputs *and* to follow this by the *sgn* function. I have left out all arcs where the weight is zero, and also the constant in the input layer, because it looked lonely with no arcs coming from it.

Figure 5.10: A Four-Layer net for generalised XOR



Now suppose I give you as input to the net a point in the first quadrant, that is, both x and y are positive . Then A outputs -1, B outputs +1, C outputs +1 and D outputs -1. E and F receive input sum zero and with negative threshold output -1, G has two -1 inputs which overcome the weight of +1, so the output from the net is -1.

If I input a point in the second quadrant, x is negative and y is positive. So both A and B output +1, while C and D output -1. E outputs +1 and F outputs -1. With that input and a positive `threshold', G outputs +1, the output from the net. If the input has both x and y negative, from the third quadrant, then as in the first quadrant, E and F output -1 and the net output is also -1. Finally, in the fourth quadrant where x is positive and y is negative, C and D output +1 while A and B output -1, so E outputs -1 and F outputs +1, hence the net outputs +1. So for input in the first or third quadrants, the unit outputs -1, while in the second and fourth it outputs +1. In other words it solves the generalised XOR problem.

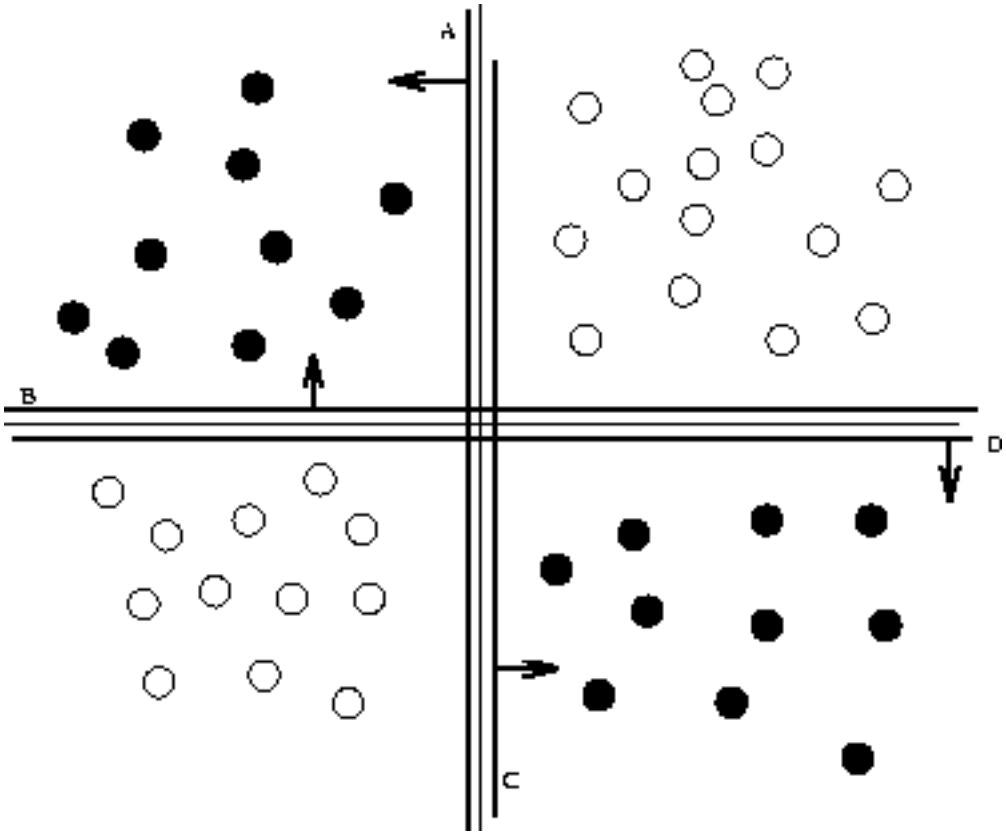
Inspection of the net and a little reflection shows how this is accomplished. It is useful to draw the lines in the plane implemented by the first layer units; they are indicated, somewhat displaced from their true positions in *Fig.4.11.* A and B return positive for the second quadrant, C and D for the fourth quadrant. Unit E is an *AND* gate which fires (+1) if and only if the input is in the second quadrant, unit F is and *AND* gate which fires iff the input is in the fourth quadrant. Unit G is an *OR* gate which fires if E fires or if F fires. It is easy to see that AND gates and OR gates are arranged (for any number of inputs) by judicious choice of the threshold. The effect of this is to partition the space so that we no longer have the extreme annoyance of having to put up with hyperplanes that run right across the whole of it.

Some more reflection makes it plain that it is easy to take any data set and chop it up into regions which are bounded by convex polytopes and where each region contains points of only one category. This holds in any dimension and for any number of categories.

Thus we conclude several things; first that four layer nets (three layers of processing units) suffices for any data set with the advantage over three layer (two layers of processing units) nets that the number of units required is of order the complexity of the data set, not the number of points, where the complexity bears some relationship to the geometry. This means that we do not feel any particular attraction towards nets with many more layers. The simpler models are more appealing to anyone who has read the last chapter thoughtfully.

Second, that it is tempting to fix the weights on all but the first layer, and to tackle a problem with some fixed number of convex sets ready to wrap around the regions. It is true that we do not know in advance whether we can do the job with a given number of units arranged in a particular configuration, but that is the nature of the beast. We never know, in general, whether there is a solution space for our net. There is a trade off: if we allow the weights between the second and third layers to change, we have more flexibility in the matter of the number of units making up any polytope, and if we allow changes in the weights generally we may have more flexibility in the way the gates work, but flexibility is not always a virtue. Worms are flexible, but cockroaches can run faster. Jelly is flexible, but not the preferred medium of choice for building houses unless you are very young indeed. There is a case for having articulated rigid parts. It is found experimentally that it is possible to train an articulated neural net ten times as fast as a flexible one on problems of low complexity. The flexible ones tend to fail to converge at all in many cases. If you contemplate lines hurtling over the initial space, the next layer doing the same with the results of the output of the lines in a higher level space, and so on, you are probably inclined to find this unsurprising.

Figure 5.11: Four layer solution for generalised XOR



In conclusion, both three and four layer nets using a step function after a weighted summation of the input values, can classify any finite data set, but the three layer net will, in the case where the data set has non-trivial complexity, that is to say where it does not fall into bands which extend across the space, need a number of units which is of order the number of points, while the four layer net is able to do so with a number of units which are of order the *simplicial complexity* of the data set, where the simplicial complexity is defined as the smallest number of simplices which can enclose the data in such a way that inside each simplex the data points have the same category. Approaching a classification problem with the AND and OR weights all fixed can speed up convergence by an order of magnitude or more when the problem is solvable by such a net. We remind you that visual inspection of the data by a projection program can aid in deciding what class of net to use, how many convex regions are likely to be necessary and how complex each convex region may need to be. Such simple methods can reduce enormously the amount of time spent making essentially random hops around a huge weight space, a practice which has in recent times used up, one hesitates to say `wasted', almost as much computer time as playing arcade games.

Next Up Previous Contents

Next: Building up functions **Up:** Committees **Previous:** Capacities of Committees: generalised *Mike Alder*
9/19/1997

Building up functions

Any net implements a function from the space of inputs to the space of outputs. A single unit in $\mathbb{R}^2$, as in *Fig.5.4* is a map from $\mathbb{R}^2$ to S^0 which sends the point $\begin{pmatrix} x \\ y \end{pmatrix}$ to the value $sgn(ax+by+c)$. A three layer net such as *Fig.5.6* implements a function

$$sgn(u*sgn\,(a_1\,x + b_1\,y + c_1) + v*sgn(a_2x + b_2\,y + c_2) + w*sgn(a_3x + b_3\,y + c_3))$$

and it is easy to see that this is the weighted composite of functions implemented by each unit in the first layer of processing elements. This sort of thing can be carried on for as many layers as you like, although don't get too carried away just yet.

The data set constitutes a sort of function; it may be taken to be a function defined on the finite set of points and having values in S^0. And a neural net is not just *one* function from $\mathbb{R}^2$ to S^0, it is an infinite family of them, parametrised by all those weights. In other words, there is a manifold of functions, in general from $\mathbb{R}^n$ to S^0, and a data set, a function defined on a finite subset of $\mathbb{R}^n$ taking values in S^0, and the problem of training a data set is to find one of the functions which agrees with the data set function. The data set functions we consider in pattern recognition are binary valued of course, and the convention I use here is that they take values in S^0, although there are other applications where we take real valued functions. I shall not treat such cases on the grounds that you are already getting good value for money and if you have too many things to think about you might get depressed, but this section and the next two comprise a brief aside on the issues involved.

Any training procedure can be viewed as a process of starting off with some arbitrary function from the family and moving about the weight space until we get agreement, until we fit the data. In general we have no guarantee that there *is* such a function in the family. The XOR problem with one unit is an example. Here one must think of fitting step functions defined on the plane, $\mathbb{R}^2$, and taking values ± 1. It is easy to see that for three or four layer nets of the sort we have described, the complexity of the data set, however defined, is the critical consideration in whether or not the problem is solvable at all. All problems have *some* size net which can solve it; equally, all nets have some problems they cannot solve. This is not a particularly profound observation, but it raises the ante from a sort of Minskian disenchantment with the capacity of the net to other more interesting issues.

If I put the problem in the frame of function fitting from a known family of functions, then I raise immediately the question of whether the family is a good one to try to fit from. The question of what

makes a sensible basis of functions for describing point sets came up in chapter two, where we were trying to describe characters, and we meditated on the merits of the Fourier basis of sines and cosines against polynomials. For a given type of data, some basis functions make more sense than others. Just choosing a basis set because it is there is not particularly intelligent; it may be that choosing it because it is easy to do the sums with it makes sense, or it may be that the kinds of functions you want to fit are likely to be well suited to the family you want to fit with. But a little thought can save a lot of computing time.

The geometry of a neural net determines the function class from which you are trying to fit the data. This function class is the important thing, and the net diagram is otherwise largely uninformative, although it may make the programming task of evaluating it somewhat simpler. It determines the dimension of the space of weights through which you are proposing to step in order to find a solution. Stepping blindly may be the best you can do, but it is to be avoided if possible. It is clear that the time to find a solution goes up as the dimension of the search space, and rather worse than linearly in general. There is a trade off between finding, on the one hand that there is no solution with your net because it isn't big enough, against at least discovering, on the other hand, this uncomfortable fact fairly quickly. Bunging everything into a huge net, running it on a supercomputer and leaving it to chug away for a few weeks (or years) is the sort of cretinous stupidity one associates with....., well, not with engineers, I hope.

Next Up Previous Contents

Next: Smooth thresholding functions **Up:** Committees **Previous:** Four Layer Nets *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Back-Propagation **Up:** Decisions: Neural Nets(Old Style) **Previous:** Building up functions

Smooth thresholding functions

- Back-Propagation
- Mysteries of Functional Analysis
- Committees vs Back-Propagation

Mike Alder
9/19/1997

Back-Propagation

The suggestion that I offered above for modifying the state of a committee by adapting all the wrong units by different amounts, more if they are less wrong, has some common sense justification, and also empirical justification: it works. Those of an idealistic cast of mind find such justifications unsatisfying, they wish to derive practice from more fundamental principles. Always nice, provided it gives the right answers, says the engineer to himself.

The approach taken by Anderson, Rumelhart, Hinton and Williams was to replace the *sgn* function by a smooth approximation to it. The function $\tanh$ is favoured for reason which are not altogether compelling; it is common to use thresholding systems which output values to or 1 instead of -1 or +1, in which case the sigmoidal approximation becomes the function:

$$sig(x) = \frac{e^x}{e^x + e^{-x}} = \frac{1 + \tanh(x)}{2}$$

It should be noted that it doesn't make the slightest difference whether you prefer 0 to -1 for representing the other state or not. If you have a predilection for 0 and 1 , add one to my numbers and divide by 2.

This smooth approach has a considerable advantage from the point of view of the idealist who wants a rule for arbitrary nets. It means that the function implemented by the net is now a differentiable function, not only of the data, but also of the weights. And so he can, when the outcome is wrong, find the direction of change in each weight which will make it less wrong. This is a simple piece of partial differentiation with respect to the weights. It can be done for any geometry of net whatever, but the layered feedforward nets make it somewhat easier to do the sums.

What happens may be seen in the case of a single unit, where the function implemented is

$$n(x,y) = sig(ax + by + c)$$

which is a composite of the function $A(x,y) = ax + by + c$ with the function

$$sig(t) = \tanh(t)$$

Now partially differentiating with respect to, say, a, gives

$$\frac{\partial n}{\partial a} = x\, sig'(ax + by + c)$$

and similarly for the other weights.

If we choose to adopt the rule whereby we simply take a fixed step in the direction opposite the steepest gradient, the steepest descent rule, then we see that we subtract off some constant times the vector

$$sig'(ax + by + c) \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

from the weight vector $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$

which is the step rule variant of the perceptron convergence rule, multiplied by the derivative of the sigmoid evaluated at the dot product of the weight vector with the augmented data vector. A quick check of what you get if you differentiate the sigmoid function reveals that this function has a value of 1 at the origin and decreases monotonically towards zero. In other words, it simply implements the differential movement rule suggested for committees, the further away you are, the less you move. If we use $\tanh$ as our sigmoid, the derivative is $\frac{4}{(e^x+e^{-x})^2}$ which for large x is approximately $4e^{-2x}$

In the case of the three layer net, what I have called (following Nilsson) a committee net, the function implemented is

$$n(x,y) = sig(u * sig(\mathbf{a}_1 \cdot \hat{\mathbf{x}}) + v * sig(\mathbf{a}_2 \cdot \hat{\mathbf{x}}) + w * sig(\mathbf{a}_3 \cdot \hat{\mathbf{x}}))$$

where *sig* is the sigmoidal approximation to the *sgn* function and $\mathbf{a}_i, i = 1,2,3$ are the three weight vectors. $(\mathbf{a}_1 \cdot \hat{\mathbf{x}})$ is just $(a_1 x + b_1 y + c_1)$

If we partially differentiate this with respect to, say, a_1, we get

$$\frac{\partial n}{\partial a_1} = sig'((u * sig((\mathbf{a}_1 \cdot \hat{\mathbf{x}})) + v * sig(\mathbf{a}_2 \cdot \hat{\mathbf{x}}) + w * sig(\mathbf{a}_3 \cdot \hat{\mathbf{x}})) * u * sig'(\mathbf{a}_1 \cdot \hat{\mathbf{x}})x$$

and similar results for the other parameters. Taking a step against the gradient means decreasing a_1 by some constant times this number, and corresponding amounts for the other parameters.

This is easily computed from the output backwards. If we use $\tanh$ as the sigmoid, we have the useful result that the derivative is $1 - \tanh^2$, and this means that no further calculations of transcendental functions is required than was

needed for the evaluation. The generalisation to feed forward nets with more layers is straightforward. Turning the idea of gradient descent (pointwise, for each data point) into an algorithm is left as an easy exercise for the diligent student. Alternatively, this rule is the *Back-Propagation Algorithm* and explicit formulae for it may be found in the references. More to the point, a program implementing it can be found on the accompanying disk.

It is noticeable that the actual sigmoid function does not occur in any very essential way in the back-propagation algorithm. In the expression for the change in a_1 for the three layer net,

$$\frac{\partial n}{\partial a_1} = sig'((u * sig(\mathbf{a}_1 \cdot \hat{\mathbf{x}}) + v * sig(\mathbf{a}_2 \cdot \hat{\mathbf{x}}) + w * sig(\mathbf{a}_3 \cdot \hat{\mathbf{x}}))u * sig'(\mathbf{a}_1 \cdot \hat{\mathbf{x}})x$$

(where we have to subtract some constant times this) the value of $sig(a_1x + b_1y + c_1)$ lies between ± 1, and if the sigmoid is a

steep one, that is to say if it looks like the *sgn* function, then the difference between *sig* and *sgn* will not be large. The derivative however does have an effect; the closer *sig* gets to *sgn*, the closer the differential committee net algorithm based on the derivative of *sig* gets to just moving the closest unit.

Next Up Previous Contents

Next: Mysteries of Functional Analysis **Up:** Smooth thresholding functions **Previous:** Smooth thresholding functions *Mike Alder*

Mysteries of Functional Analysis

Once one has taken the step of extending finite maps having values in S^0 by smooth maps into [-1,1], the smooth maps being from some manifold of such functions, the subject starts to attract the interest of Functional Analysts, a collection of Mathematicians who, having reduced much of the theory of functional approximation to linear algebra, find themselves underemployed and reduced to manufacturing rather artificial problems, those real problems that are left being much too hard. There have been a number of theorems showing that any function from $\mathbb{R}^n$ to [-1,1] which is sufficiently well behaved (for example, continuous and having compact support) may be approximated arbitrarily well by three layer neural nets. Such statements are true but of very limited practical utility. They are not offered to the engineer because they have practical utility, but because they are publishable, and they are publishable because engineers who are not familiar with a branch of mathematics are more likely to be impressed with it than they are with familiar tools. It is desirable therefore that the elements be explained in order to reduce the reader's gullibility quotient.

Whenever one talks of approximating one thing with another, one has some kind of a notion of a distance between the things, and this distance being small. More precisely, to be told that one can approximate a function arbitrarily well by polynomials is to be told that there is a sequence of polynomials, and that the further you go down the sequence, the smaller the distance between the polynomial you choose and the function you are getting closer to. And in fact the function you are approximating has to be the limit of the sequence of polynomials; the distance between polynomials in the sequence and the function you are trying to sneak up on can be made as small as anyone wants just by going far enough down the sequence. This leads us to ask how you measure the distance between functions. Suggesting, as it does, that there is one and only one `right' way, this is actually the wrong question, but only a bit wrong.

There are many ways of measuring the distance between functions, but two are of particular importance. The first is the so called L_∞ norm, which measures the distance of f from g by taking

$$\|f - g\|_\infty = \sup\{|(f - g)(x)|\}$$

If f and g are continuous and the domain of both is compact, then this is always defined. The second is the so called L_2 norm, in which the distance of f from g is obtained by taking

$$\|f - g\|_2 = \sqrt{\int (f(x) - g(x))^2}$$

This is defined in the case where f and g are continuous and the support is compact, but in much more general cases also. If you don't know what *compact* means, the excellent book by G.F. Simmons given in

the references at chapters end will tell you, along with much more material worth knowing if only so as not to be intimidated by it. Although the numbers come out as different for any two functions, the two norms are *equivalent* on the smaller space of continuous functions having compact support in the sense that if you have a sequence of functions approaching a limit function in the one sense of distance, then the same sequence will also be converging to the same limit in the other sense too. More bizarre possibilities exist, so watch out, but for the cases discussed, the different norms are not too different.

In either sense, it is often useful to be able to find out what functions can be expressed as limits of some class of functions. If, for example, we take functions on the real line which are *pdf*s, i.e. they are non-negative and have integral 1, then it might be cheering to be able to prove rigorously the statement I threw off rather casually some time back, that such a thing can be approximated as well as you wish by a mixture of gaussians. To do this, you need a careful definition of what it means to approximate a function *g* by some family f_p, and of course it means I can choose members of the family $f_1, f_2, \cdots f_n, \cdots$ such that

the distance $\|f_n - g\|$ can be made less than any given number ε by choosing *n* large enough. Thus, it is easy to prove that in the space $\mathcal{C}^0([0,1])$ of continuous functions from [0,1] to $\mathbb{R}$ with the L_∞ norm, the family of polynomials can be used to approximate any continuous function. This is a celebrated theorem of Weierstrass.

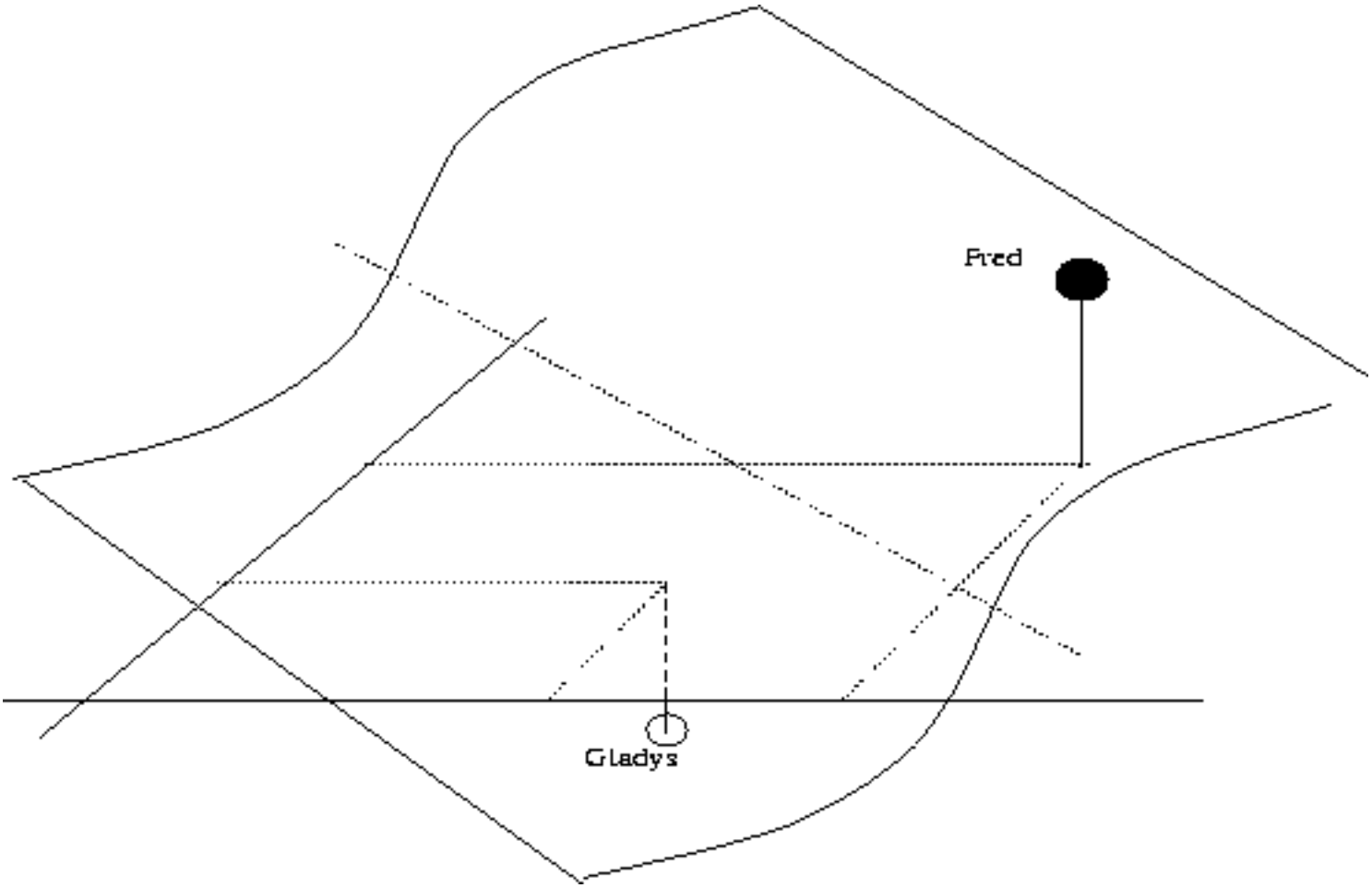
The theorem of Weierstrass must be compared with the well known Taylor expansion of a function about a point, which for the case of infinitely differentiable functions actually gives you a polynomial and bounds on the difference between the function and the approximating polynomial. Sometimes the bounds can be estimated, which makes for a really practical theorem. You can replace one, possibly horrid, function by a polynomial *and work out the worst case discrepancy*. The Weierstrass theorem works for a larger class of functions, but doesn't tell you how bad or good a particular approximation is. (Although there are constructive proofs which *do* give the information. Functional Analysts often seem quite indifferent to these practicalities, alas.)

Moving away from the real line into $\mathbb{R}^n$ and even more awful generality, the generalisation of Weierstrass' theorem which appeals to Analysts is the *Stone-Weierstrass Theorem* which may be used to show that if any family of functions on a compact domain is an *Algebra* of continuous functions, which means that the sum of functions in the family is in, the scalar multiple of any functions in the algebra is in, and the product of any functions in the algebra is in the algebra, and if there are enough functions to separate distinct points (i.e. if there are two distinct points *x* and *y* in the domain of the algebra functions then there are functions *f* and *g* such that $f(x) \neq g(x)$) and if, finally, the algebra contains the constant functions, *then* any continuous function with the same (compact) domain can be approximated as closely as you like in the L_∞ sense. Well, a little work has to be done to bend the Stone-Weierstrass

theorem to make it go, but it can be used to show that for compact subsets of $\mathbb{R}^n$, any continuous function can be approximated as closely as you like by the functions obtained using three layer neural nets with any continuous sigmoid function.

It is worth drawing some pictures here to see what is being said. If you take a square in $\mathbb{R}^2$, any single unit will implement a function that is just a smoothed version of the function that assigned +1 to Fred and -1 to Gladys, with a dividing line down in between them that went right across the whole plane. *Fig.5.12* tries to give an incompetent artists impression of what the function looks like, a sort of Wave Rock effect.

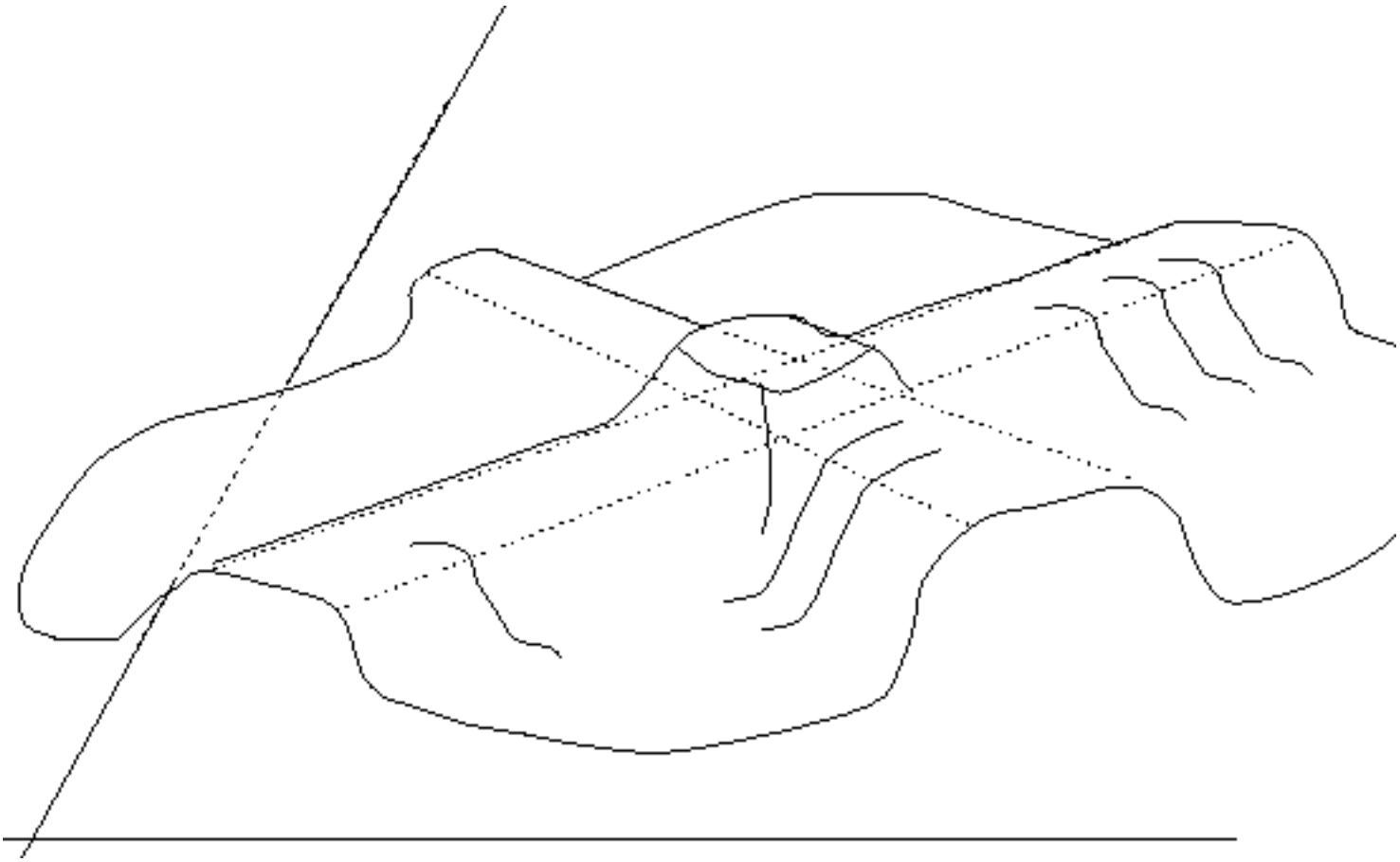
Figure 5.12: A sigmoid classifying.



Now putting three of these in at the first processing layer of a three layer neural net, and then following them with one output unit, means that we can get scaled multiples of translates of things like the Wave Rock . We can even turn it upside down by multiplying by a negative number. So we have to add the result of stretching such a thing out in the x-y plane and then multiplying its height by some number and adding to the result of doing the same thing to two other such functions. It is clear that the result must have three waves reaching right across the plane. If the lines are parallel, we could use two of them to

make an infinitely long ridge. Four of them could make two ridges with a sort of tower where they cross. But the waves go all across the plane. Just as we politely rejected the three layer committee net for the generalised XOR data set, on the grounds that a model where the complexity goes up with the amount of data, leaves something to be desired, so we should be rather inclined to abandon the three layer smoothed version as a suitable basis for modelling functions, unless the behaviour whereby there are waves which extend across the space were a feature of the functions we wanted to model. The analysts claim that it can always be done is correct, but the implied claim that it is worth doing is deeply suspect. The analysts will cheerfully explain that they never said it was a good idea to actually do it, their theorems only say it can be done. The engineer will reply that they should, in that case, publish these results in some Pure Mathematical journal dedicated to useless trivia. This is an argument that can go on a long time. The moral is that although the analysts and other pure mathematicians are all God's creatures, and although they often say things that are of interest, one should read what they say carefully and very, very literally. They are not necessarily on your side.

Figure 5.13: Sigmoids summing.



Of course the one dimensional case is quite useful since it can be used to give a quick proof that any *pdf* can be approximated by gaussians, as you can use half a gaussian turned upside down and glued to the other half as a (sort of) sigmoid.

The theory of Fourier Analysis and the Fourier series whereby we build up a function out of sines and

cosines extends the Weierstrass result in a certain sense to the case where we may use the L_2 norm, and hence to a larger class of functions. The theory can be modified to give results for three layer nets, but such results are no more practical than the Stone-Weierstrass Theorem. The question of interest is, how fast does the discrepancy disappear with the number of units used? Contemplation of *Fig.5.13* suggests that discrepancies are likely to be of the order of $1/m$, where m is the number of units This does not compare well with, say, Taylor's theorem for most analytic functions encountered in practice. Analysts are not usually interested in what functions are actually encountered in practice, and it is as well to remember this. Mathematics is a splendid servant but a thoroughly foolish master and it is essential to let it know who is the boss.

What we did with four layer nets and the hard limiter *sgn* function can be done with smooth sigmoids and function approximation, if you should feel a need to approximate functions. Since reality always gives you finite data sets to finite precision when you measure it, the need is seldom irresistible, although there are many circumstances when the procedure is useful.

Next Up Previous Contents

Next: Committees vs Back-Propagation **Up:** Smooth thresholding functions **Previous:** Back-Propagation

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Compression: is the model **Up:** Smooth thresholding functions **Previous:** Mysteries of Functional Analysis

Committees vs Back-Propagation

It is natural to wonder whether there are any advantages in using Back-propagation over a committee. The committee is experimentally found to be faster than a three layer net by about an order of magnitude and either converges fairly quickly or never converges because the problem is unsolvable by the given committee. The three layer back-propagation net sometimes converges and sometimes doesn't, depending on the initial state. There is a theorem which justifies the back-propagation algorithm, sort of, but no actual proof of convergence. The state space is bigger for back-propagation, which gives more flexibility, but flexibility is a property of jellyfish and earthworms which hasn't done them a lot of good when it comes to arguing the toss with insects and vertebrates. There are more of us than of them. So one should not be unduly influenced by such simple arguments as the unbridled advantages of the larger state space.

Experiments suggest that the so called three layer net has no practical advantages over a committee of slightly larger size and an equal number of parameters, and indeed is likely to take longer to converge on average. It is fairly easy to understand the thinking behind a committee net, while back-propagation tends to numb the mind of the average user for whom calculus of more than one variable has essentially the status of black magic in the wilder parts of New Guinea. Unfortunately, superstition is rife these days, and the incomprehensible is thought to have *manna* not possessed by the intelligible.

For four layer neural nets, the articulated four layer net with a fixed number of units in the second layer partitioned into groups intended to cooperate with each other in surrounding a convex region of points of some type, the partitioned groups then being ORed with each other to give a decomposition of the data as a union of convex regions, again outperforms the four layer back-propagation experimentally, being about an order of magnitude faster on sets of moderate complexity with greater relative improvement on more complex sets. Mind you, this is a competition between losers, there are better ways. It is true that sometimes back-propagation can eventually find a solution which the articulated net cannot, but at least with the articulated net you find out that there is no solution with your given geometry rather faster.

Next Up Previous Contents

Next: Compression: is the model **Up:** Smooth thresholding functions **Previous:** Mysteries of Functional Analysis *Mike Alder*
9/19/1997

Next Up Previous Contents

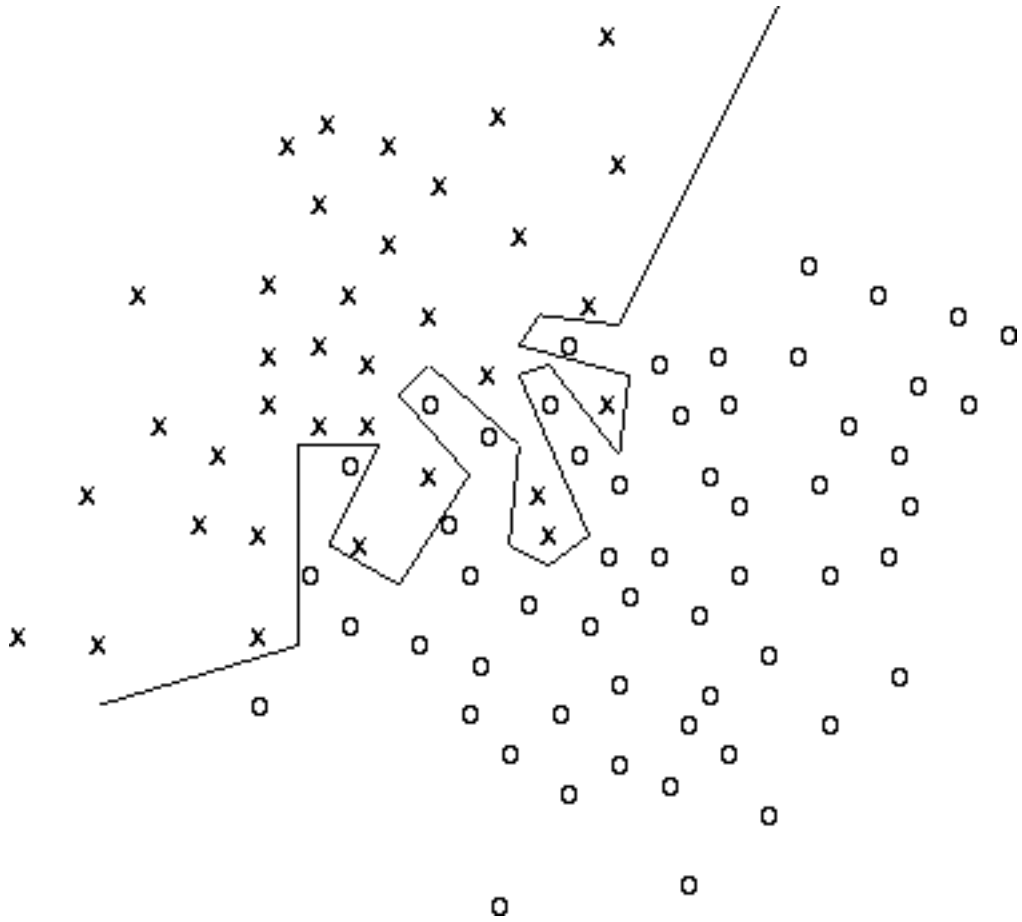
Next: Other types of (Classical) **Up:** Decisions: Neural Nets(Old Style) **Previous:** Committees vs Back-Propagation

Compression: is the model worth the computation?

The purpose of using neural nets is to try to extract the maximum amount of information about the category of the data, on the hypothesis that there will be some more points along soon and we want to make the most intelligent guess at their category. If the complexity of the data is sufficiently high compared with the number of points, then the simplest neural net is worse than a simple list of the points and their categories. It is true that such a procedure gives no suggestion as to what to make of the category of the next point along, on the other hand if an attempt to find stucture in the data has failed, then we might as well guess. And a data set which has shown no compression using a neural net is one where guessing merely confronts the fact that we don't have any idea of what the category of a new point is, rather than permitting us to deceive ourselves by using a neural net.

Imagine that we have a data set such as the male and female height-weight points of chapter one. It is possible to visualise an immense amount of data and to be able to believe that two gaussians, one for each category, fit the data pretty well out to three or four standard deviations. Now a four layer neural net which got every point correct would be hard to believe, and a three layer neural net utterly impossible to take seriously. All those lines which carefully snuck through the enemy camp would lead you to expect points of each category in the most peculiar places. How can we articulate our natural scepticism using Rissanen's ideas?

Figure 5.14: An implausibly complicated theory.



If we add to each datum in $\mathbb{R}^n$ its category regarded as a number either ± 1, then it takes one extra bit to do a binary classification, over the bits needed to say where the datum is in the space. Similarly, if we have gaussians doing a classification, we need to add in a bit to say which category of gaussian they are supposed to be.

In order to specify a single unit in $\mathbb{R}^n$ we need, on the face of things, $(n+1)P$ bits, where P is the precision used to specify each coefficient. It is slightly better than this in practice; multiplying the $n+1$ numbers by a positive constant will not change the output of the net or the position of the line in any way; we might therefore stipulate that the coefficients be normalised, say by making the sum of the squares of them equal to 1. This ensures that the coefficients or weights always lie between ± 1, and that if we have n of them we can calculate the last. We therefore need nP bits to specify a single unit in the second layer.

If n = 2 and P = 10, we specify the line with coefficients which are accurate to one point in a thousand. This is often excessive, since the amount of wobble of the line which can be applied and still have it implement the dichotomy may be quite large. To analyse the situation, suppose we have a binary data set in the unit square with points of category +1 at values of x and y greater than 0.6 and points of category -1 with both x and y less than 0.3. Then a specification of the position of the line needs only about three bits precision on each coefficient, if the coefficients are chosen to be between +1 and -1. To specify the data set takes some number of bits to specify the positions, say NP, and to specify the category takes a

further *N* bits when there are *N* points. To use a line instead for the specification of the category, takes 6 bits, three for each (normalised) coefficient. So provided there are more than 6 data points, we have made a saving by using the line. More generally, we have that the number of points needed in dimension *n* in order to justify using a hyperplane is *nP*, where *P* is the precision required. If the points form two clusters which are well separated, then *P* goes down, which seems reasonable. There is no attempt to specify the location of the data by the model in this analysis, we only use it to store the category. If we use a committee net with k units to store the category for a binary data set in $\mathbb{R}^n$, then we have *knP* bits required to specify the state of the net, and again N bits for the data. If we have m categories instead of just two, we need $N \log_2(m)$ bits for the data set. In addition, there is some overhead needed to specify the network topology.

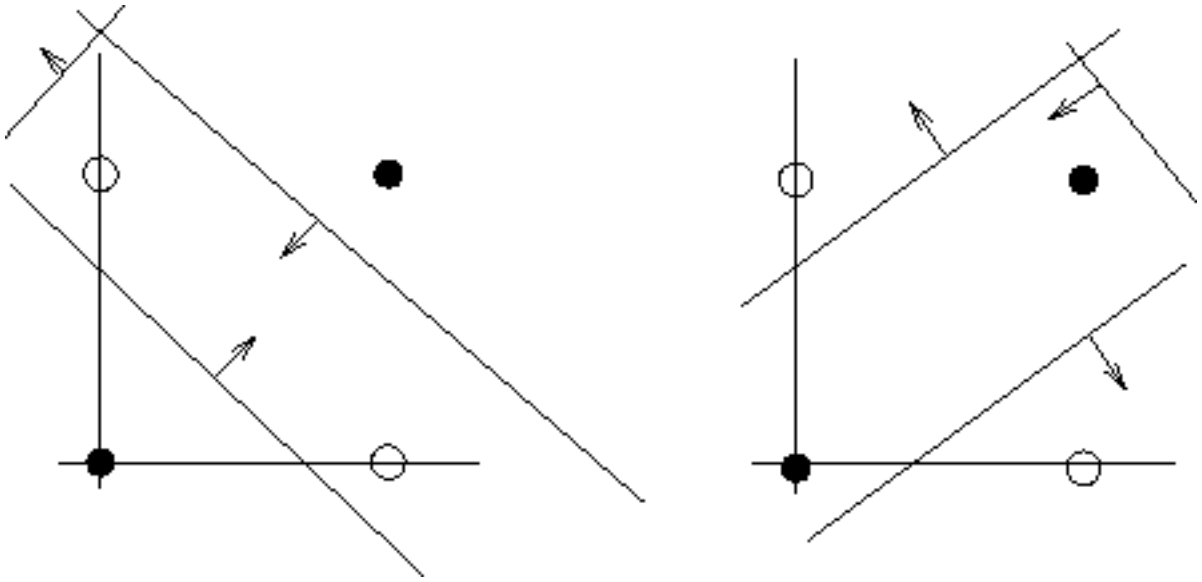
A further problem is that we do not know in advance what the precision of the coefficients of the net should be: we can put a bound on this by pointing out that it is senseless to have the precision of the coefficients much greater than that of the data, but the precision of the data is largely irrelevant in the model I propose. It is infeasible to go much further without some delicate arguments which might be thought tendentious, but it is clear that a neural net having *k* nodes in $\mathbb{R}^n$ must classify at least *knP* binary data points, for *P* at least of order 10, before it can be said to be extracting any structure from the data set. Papers discussing the classification of data in dimension 100 with nets having hundreds of units must therefore have about half a million data points if the application is not to be an exercise in self-deception. This criterion would exclude some well known papers in the literature. Since P is often 64 bits, and the specification of the network topology may be of order $k(k-1)$ bits, this is an optimistic analysis of the state of affairs. And there are papers in which thousands of units in dimension several hundred have been cheerfully applied to data sets having only a few hundred points in them. The naive belief that in doing this the protagonists were contributing anything to the advance of human knowledge, would appear to be sadly in error. Had they spent the computer time playing *Super Mario Brothers* they might have had more fun and not wasted the resources any more flagrantly. The figure of 10 for P agrees with rules of thumb used by the neural net community for some time, but there has been little rationale offered until recently.

It is important to understand then that when you read a paper describing a neural net application in which there are fewer than *knP* binary data classified by a net, the writer has not used his net to extract information from the data. On the contrary, he has put information in. The extra information is not about the data, it is about the deficiencies of his education.

It is very hard to believe that a model of a process which has failed to extract any information from the data given, will have any real capacity to predict the results of more data measurements. In the case of neural nets which, like Feed-forward Back-Propagation nets, start with some random initialisation, then there is not, in general, any unique solution. Moreover, the less information is extracted from the data in finding a solution, the larger the range of solutions there will be. Now two different solutions will disagree with each other in their predictions, by definition, and the larger the space of different possible solutions, the greater will be the extent to which solutions can disagree with each other about the result of classifying a new datum. We cannot therefore feel any confidence about any one prediction which such a net provides, because we know that there are other solutions, just as compatible with the data, which yield other, contradictory, predictions.

In the XOR case, for example, we have four data points and three units in a committee which can solve the problem.

Figure 5.15: Two committee solutions for XOR.



If we consider the two solutions of the committee net shown in *fig.* 5.15, we see that they differ about the state of a point in the middle, as well as at other regions. Now which solution you get will depend on the initialisation of the weights, and the random sequence of calling data points. So if you ask for the category of a point in the middle at coordinates (1/2,1/2), you will find that the answer supplied by the net will depend on the random initialisation. In other words, you are consulting a rather bizarre random number generator. It would be simpler to toss a coin. Instead of initialising a coin with an unknown starting velocity and momentum, one has initialised a back-propagation network with random values, and the resulting prediction for a new datum depends on just which initialisation was chosen. Coins are quicker, cheaper and no less likely to give the right answer.

Anyone who has any kind of model fitted to data where there are less data than parameters to be fitted, is deceiving himself if he imagines that his model has any useful predictive power, or indeed any value other than as an indicator of incompetence. All he is doing is the equivalent of simulating a coin toss, with the drawback that it takes much longer and wastes a deal of computer time.

Next Up Previous Contents

Next: Other types of (Classical) **Up:** Decisions: Neural Nets(Old Style) **Previous:** Committees vs Back-Propagation *Mike Alder*
9/19/1997

Other types of (Classical) net

- General Issues
- The Kohonen Net
- Probabilistic Neural Nets
- Hopfield Networks
  - Introduction
  - Network Characteristics
  - Network Operation
  - The Network Equations
  - Theory of the Network
  - Applications
- The Boltzmann Machine
  - Introduction
  - Simulated Annealing
  - Network Characteristics
  - Network Operation
  - Theory of the Network
  - Applications
- Bidirectional Associative Memory
  - Introduction
  - Network Characteristics
  - Network Operation
  - The Network Equations
  - Theory of the Network
  - Applications
- ART

- Introduction
- Network Characteristics
- Network Operation
- Theory of the Network
- Applications

- Neocognitron
  - Introduction
  - Network Structure
  - The Network Equations
  - Training the Network
  - Applications
- References
- Quadratic Neural Nets: issues

Mike Alder
9/19/1997

Next Up Previous Contents

Next: The Kohonen Net **Up:** Other types of (Classical) **Previous:** Other types of (Classical)

General Issues

It is worth contemplating again the nature of the problems which can usefully be tackled by neural nets. As mentioned in chapter one, supervised learning means function fitting. We have some quantity of data which are represented as points in $\mathbb{R}^n$ for some *n*, and we have values assigned to each of these points which may be real values or may be integers representing the categories to which the points belong. A Neural Net has to be prepared to say what the value or category is for some new point it hasn't seen before. In other words, it has to assign a value to each possible point of the space which is a potential datum. Which means it is a map or function defined on the space $\mathbb{R}^n$, or a piece of it, which agrees, more or less, with the values for the data.

Unsupervised learning, by contrast, means finding clusters in the data, deciding when two things belong in the same category without being told what the categories are. The space may be discrete, as when we are looking at 11 by 9 grids of binary pixels, the case where we give a list of different characters without saying which each is supposed to be and ask the system to pick out the different letters without knowing what they are. Or it may be a continuum. Either way we can represent the data as points in $\mathbb{R}^n$ for some *n*. And inevitably we run into problems of what `close' means. Clustering is a complicated business which can be accomplished in lots of ways, with nuances I have not discussed. But the two problems although related are distinguishable, and the basic framework is worth bearing in mind in what follows.

Unsupervised learning is also referred to in the literature as the behaviour of a *self organising system*; I suppose there is an image, invoked by this terminology, of a chemical broth evolving through amino acids to proteins to a tribe of jelly fish. Since we are interested in winding up with a program that can respond usefully to a new datum such as an image, and are not proposing to simply watch on our computer screen a sort of Conway's Life game, evolving Truth and Beauty, as a substitute for Saturday afternoon football on television, our interest in self organising systems is confined to finding clusters in the data. What is meant by a `cluster' is left deliberately vague for the moment, but classifying human beings into males and females can be done reasonably well in the height-weight diagram by noting that two bivariate gaussians seem to fit the data in a natural way. And it isn't too unreasonable to suppose that if we measure a rather larger number of parameters by looking at and listening to them, we might find two largely discriminable clusters in the larger representation space. So although we wouldn't have the names of the categories, we would have a system set up for doing a classification or recognition job by concluding that there are some natural categorisations of the data, the clusters.

There are certainly subtleties to consider at a later stage, but it is as well to focus on these two matters first when considering what a neural net of some type accomplishes. Thus life is made simpler when one works out that the Kohonen Self-Organising Map (SOM) is finding a cluster with some sort of *manifold* stucture, and the ART of Grossberg is also a cluster finding system. The nets which do associative recall have the problem of mapping new points to one of the older points or some output associated with the

older points on which the system has been trained.

Another distinction which may be of practical importance is the `on-line/off-line' classification. Is the data given all at once, or is it coming in serially, one datum at a time, or is it coming in in lumps of many data at once, but not all? Most statistical methodologies presuppose that all the data is there to be operated on in one pass, in order to do either the function fitting or the clustering, while most neural net methodologies assume the data is coming in sequentially. The neural netters therefore tend to use a dynamical process for generating a solution, something like a Kalman Filter, which starts with a possibly random initial estimate and then updates it as the data comes in down the line. Bayesian methods lend themselves to this kind of approach, and other methods can be adapted to it. This leads us to study dynamical systems which find clusters, or which fit functions to data.

In discussing neural nets in this way, as things which find clusters or fit functions from a family to data, we look at them from the point of view of their function rather than their form. Giving a network diagram while omitting to say what kind of thing the net is trying to do is not entirely helpful. This level of abstraction makes many people extremely unhappy; they feel insecure without concrete entities they know and love. Such people concentrate on details and fail to see the wood for the trees. Their explanations of what they are up to are confusing because it is hard to see what problem they are trying to solve or what kinds of methods they are using. Abstraction for its own sweet sake may be a sterile game for losers, but being able to figure out what *kind* of thing you are doing can help you do it better, and also allow you to see that two methods that appear to be different are really the same with the names changed.

Next Up Previous Contents

Next: The Kohonen Net **Up:** Other types of (Classical) **Previous:** Other types of (Classical) *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Probabilistic Neural Nets **Up:** Other types of (Classical) **Previous:** General Issues

The Kohonen Net

Inspired by some results obtained in Neurophysiological studies, where it has been shown that there is a mapping between the world and adjacent neurons in the human cerebral cortex, Kohonen in the 1980's showed how a set of neurons could plausibly be constructed so as to adapt to extract the shape of a data set. It is important to understand that it works only for data sets which lie on curves, surfaces, or in general *manifolds* embedded in $\mathbb{R}^n$. I make a brief diversion to say what an embedded manifold is.

The figure 5.16 shows something that could arise in an image. The eye picks out three distinct curves.

Figure 5.16: Three curves in the plane.

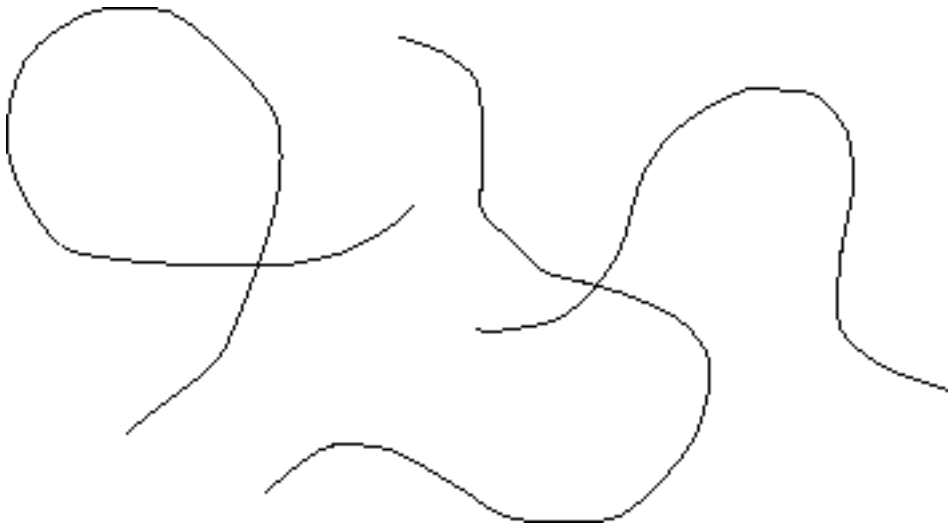
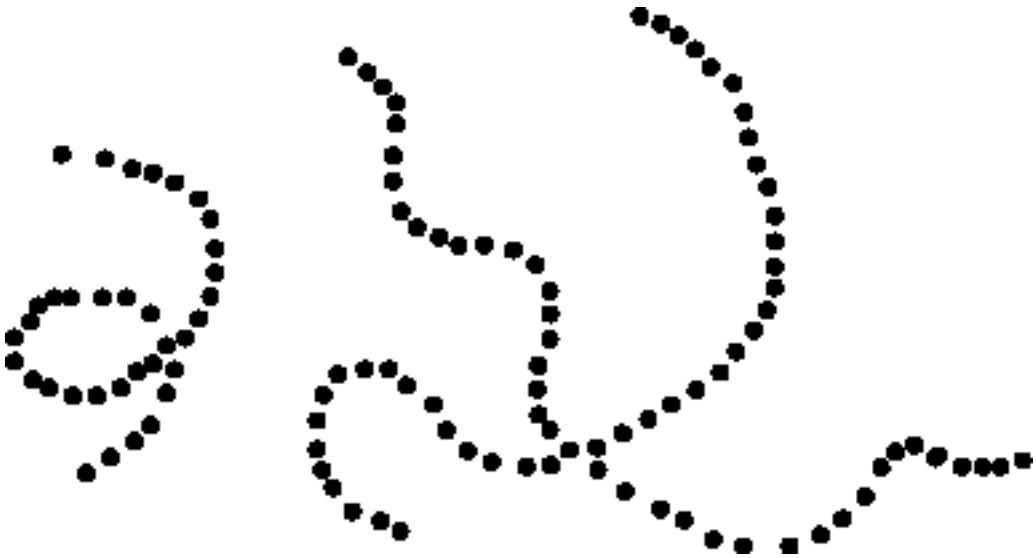


Figure 5.17: Three curves in the plane?



The following figure, 5.17, is a discrete version of its predecessor. It is most simply considered as being a finite set of points which has been drawn from three underlying continuous curves. This is closer to realism than the preceding figure, since data sets in real life are finite.

To say that there are three curves underlying the data of *figure* 5.17 is to claim that there are three `clusters' in the data. The allocation of which points belong to which of the three underlying curves is very simple to the eye, except in the case where the curves intersect, and there is only limited scope for disagreement here. So we have a kind of `cluster', picked out easily by the eye, which is very different from the sense of `cluster' which the reader may have formed after the discussion on gaussian mixture models. Nevertheless, the eye sees three distinct `objects' in the image. The points may be assigned one of three different labels.

Defining the notion of a `smooth curve in the plane' is best done by saying that it is a set of points which is the image of a twice differentiable map from the unit interval, [0,1], into $\mathbb{R}^2$. In order to ensure that it is not a space filling curve and that its existence may be inferred from a finite sample, we may stipulate that there is some bound on the curvature of any such map.

Curves are often specified in mathematics implicitly, that is to say as zeros of functions, for example:

$$\{(x,y) \in \mathbb{R}^2 : x^2 + y^2 - 1 = 0\}$$

which is the unit circle. Note that we have here a smooth map from $\mathbb{R}^2$ to $\mathbb{R}$ which imposes one condition on the two variables, thus leaving a one dimensional object, the circle.

All this also applies to curves in $\mathbb{R}^n$, and also to surfaces in $\mathbb{R}^n$. In implicit representations, we often have *k* smooth conditions on *n* variables, which may (but need not) give rise to a smooth (*n*-*k*) dimensional manifold.

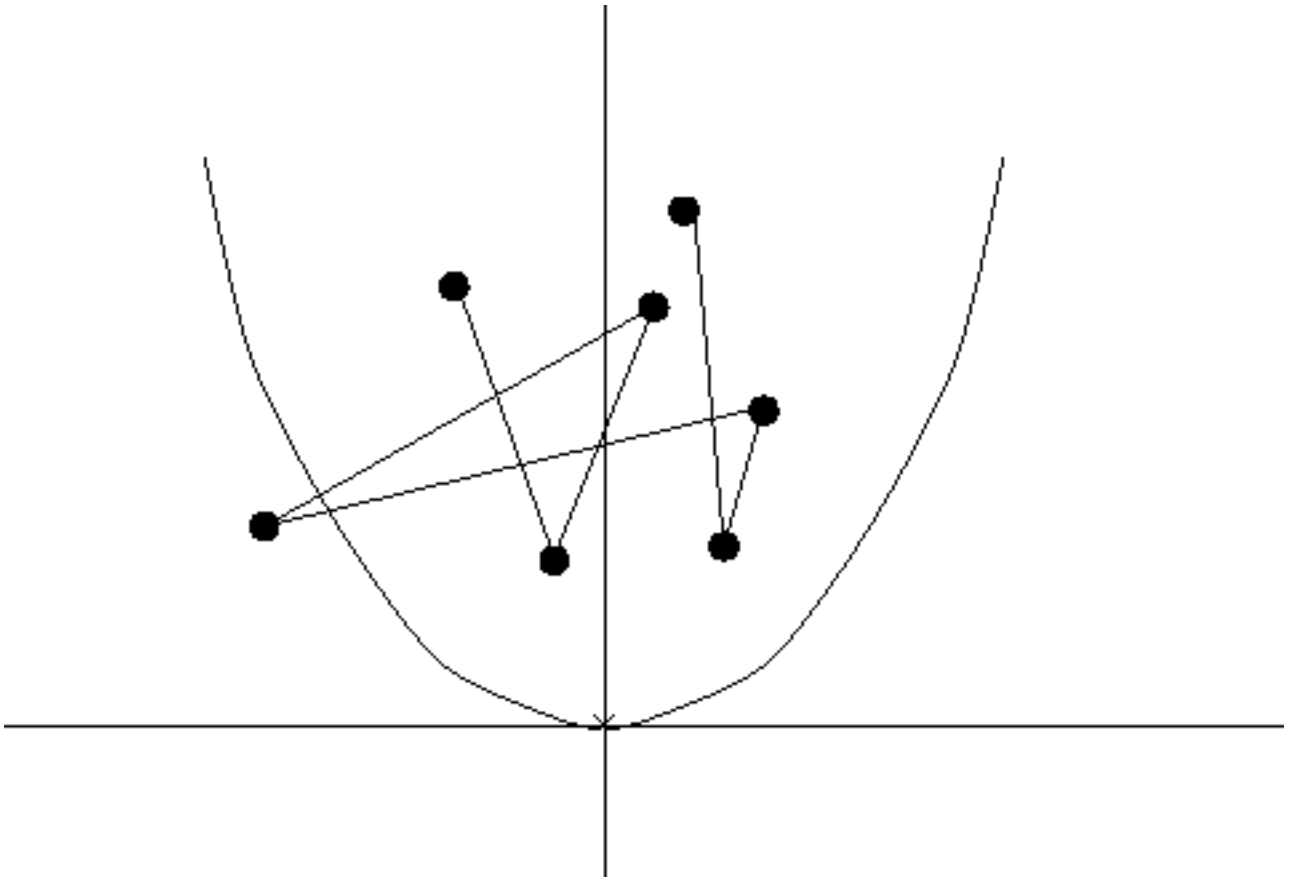
Without going into too much undergraduate geometry, if every sufficiently small open ball in $\mathbb{R}^n$ intersecting a set *U* does so in a region which can be put into a smooth 1-1 correspondence with an open

ball in $\mathbb{R}^q$, then we say that U is a q-dimensional manifold embedded in $\mathbb{R}^n$.Since there are often constraints of a physical sort on the variables which have been obtained by taking measurements of a system, it is often the case that data sets lie on embedded manifolds. For example, the vocal tract can vary, for an individual speaker, only in a limited number of possible constriction points, and this gives a two dimensional vowel space. If we measure filter bank values to get a discrete representation of the log power spectrum of speech, we may be taking 16 measurements. The vowel space for a speaker may be a two dimensional manifold embedded in this sixteen dimensional space.

The Kohonen process tries to fit a (finite) data set in $\mathbb{R}^n$ by moving a q-dimensional grid towards it. The grid can be thought of as a finite approximation to I^q, where I is the unit interval. I^q is called a *q-cube*. The following example is done in dimension one, but the idea is the same for all dimensions.

Suppose that we have a set of points in the plane that form a curve, lying, for example roughly equally spaced along the curve $y = x^2$ for definiteness. Suppose we have a line of model neurons arranged initially at random in the same space. Let the line consist of `neurons' $\{n_1, n_2, \cdots n_k\}$,where each neuron is simply a point in the space, but where the consecutive points are imagined as joined by line segments.

Figure 5.18: Initial (random) state of Kohonen network.



Present a point in the data set, x_j to the network. The Kohonen rule is to move the closest point of the set of neurons towards the calling datum, but also to move, by a smaller amount, the adjacent neurons. Adjacent not in the sense of being close in the plane, but in the sense of being neighbours in the ordering of the neurons, not at all the same thing in general. We can imagine the ordered line of neurons as being, initially, badly tangled by its random embedding in the plane.

If this process is iterated, with different choices of the data points selected to do the calling, the tangle of neurons gradually becomes straightened out and the line is attracted towards the data points and becomes an approximation to it. The two senses of `close', close in $\mathbb{R}^2$ and close in the topology of the space of neurons, become similar. Mathematically speaking, the cleanest way to look at this is to think of the tangled grid of neurons as being the image of a continuous map, f_0 from the standard *k*-cube into $\mathbb{R}^n$, or in practice from some finite approximation to such a *k*-cube consisting of a k-dimensional grid. Then a single point of the data set in $\mathbb{R}^n$ is presented to the system, and the map is modified by being made to move the points of the *k*-cube closer to the calling point, in such a way as to preserve the continuity of the map. Thus we have the whole data set acting on the continuous maps from the *k*-cube into $\mathbb{R}^n$ so as to make them fit the data set. The sequence of maps should converge to some satisfactory map eventually, if the algorithm is right.

Kohonen wrote of the `order' being preserved, although it could also be reversed by the process, if one takes it that the data set is also ordered. It is simpler to note that the mapping from the line of neurons to the line of data points is locally 1-1 or is an *immersion* in the jargon of the topologists. At least it would be if we had an infinite number of neurons which really lay along a line.

It is not difficult to generalise to more complicated objects than curves being fitted with a linear grid. If one has a set of data points in $\mathbb{R}^n$, it may be that they form a random cluster in the space, or it may be that they form a lower dimensional manifold. For example, they could be a swarm of flies all of which are sitting on a large inflated balloon; in this case the dimension of the surface of the balloon is two, less than the original three in which the flies are free to move. A fly which chooses to move only on the surface of the balloon is giving up a whole dimension, just like Obi Wan Kenobi in the *Star Wars* epic.

The set of points on the surface of a spherical balloon constitute a manifold, and the flies form a finite subset of this manifold. One can imagine linear submanifolds of $\mathbb{R}^n$ or curved ones, as in the case of the balloon. If there are enough flies, it is easy for the eye to discern the lower dimensional structure, or at least the eyes, since some parallax is necessary before it can be seen. A manifold of dimension *k* is a generalisation of the idea of a surface such as a balloon which is technically described as a 2-manifold. A *k*-manifold is said to be *parametrised* when there is some collection of k-dimensional disks which are mapped to it so as to cover it without folding. Conceptually like sticking postage stamps in order to cover a spherical balloon, such parametrisations are usually given by explicit maps, an example being the map that sends the interval $(-\pi, \pi)$ onto the unit circle by sending *t* to $(\cos(t), \sin(t))$. This covers every point of the circle except the point (-1,0), so we send another copy of $(-\pi, \pi)$ to the circle by

the map that sends *t* to $(-\cos(t), \sin(t))$,which has a lot of overlap with the first map. Each map separately is 1-1 and infinitely differentiable, we have covered or *parametrised* the circle with a pair of one dimensional disks (open intervals). A Sphere could likewise be parametrised with a couple of open 2-disks, and it is an easy exercise to find a couple of explicit maps from the interior of D^2 to $\mathbb{R}^3$ which give a parametrisation of the 2-sphere.

More generally, then, if we have a set of data points in $\mathbb{R}^n$, lying on a manifold of dimension *m*, and if we have a grid of model neurons arranged to lie in a cube of dimension *q*, the Kohonen algorithm may be easily modified to attract the grid to the data and to do as good a job of fitting the data as possible. If $q < m$ then it will resemble the case of a line of neurons trying to fit itself onto a plane, and a deal of folding will be necessary and the final fit less than adequate; if $q > m$ then it will resemble the case of a square being attracted to a curve, and a good deal of buckling will be necessary, but if $q = m$ then a relatively good fit may be obtained. The general property that the *q*-cube fits itself without local folding to a *q*-manifold may be shown to hold, although the term `order preserving' is inappropriate here. Experiments show that manifold structure can be extracted auomatically by this means. See the references for *Dimension of the Speech Space* as a case in point, and *Kohonen's algorithm for the numerical parametrisation of manifolds* for a discussion of the essential properties of the Kohonen algorithm. It is accomplishing, for a finite point set which comes from a manifold, a sort of parametrisation of part of the manifold, given by a map of the q-disk (or q-cube, or grid) into the containing space.

We can see this as a sort of structured clustering: if there is a manifold structure instead of a gaussian cluster structure this system will find it. But what if both occur? As soon as one starts to think about the possibilities for what sorts of shaped subsets of $\mathbb{R}^2$ or $\mathbb{R}^3$ we can have made up out of finite point sets, one starts to boggle a bit at the prospect of using any one system. And $\mathbb{R}^n$ for higher *n* leaves the imagination behind fast.

The general device of finding a point (e.g. a zero of a function) by making the point an attractor of a dynamical system goes back to Newton and the Newton-Raphson method. In the present case, the attractor is a set of points rather than just one, and some aspects of the topological structure is being extracted. It is a method which has some charm, and is suceptible of a range of applications; one of the few cases where an idea from the study of Artificial Neural Nets has applications outside its original domain. Chaos theorists are often presented with finite point sets in $\mathbb{R}^n$ which purport to be from a strange attractor, and trying to extract its dimension (not in general an integer, since attractors don't have to be manifolds and usually aren't) so the study of point sets and their shapes is flavour of the month in Dynamical Systems theory as well as for us humble Pattern Recognisers.

The applications to Pattern Clasification are more doubtful however, and the differences between the piecewise affine neural nets and the Kohonen nets are large. But the basic idea has novelty and is not immediately obvious. There are lots of ways of extracting linear or affine strucure in data, but when it is non-linearly arranged in the space, there are few methods available and the Kohonen process is one of them.

Next Up Previous Contents

Next: Probabilistic Neural Nets **Up:** Other types of (Classical) **Previous:** General Issues *Mike Alder 9/19/1997*

Next Up Previous Contents

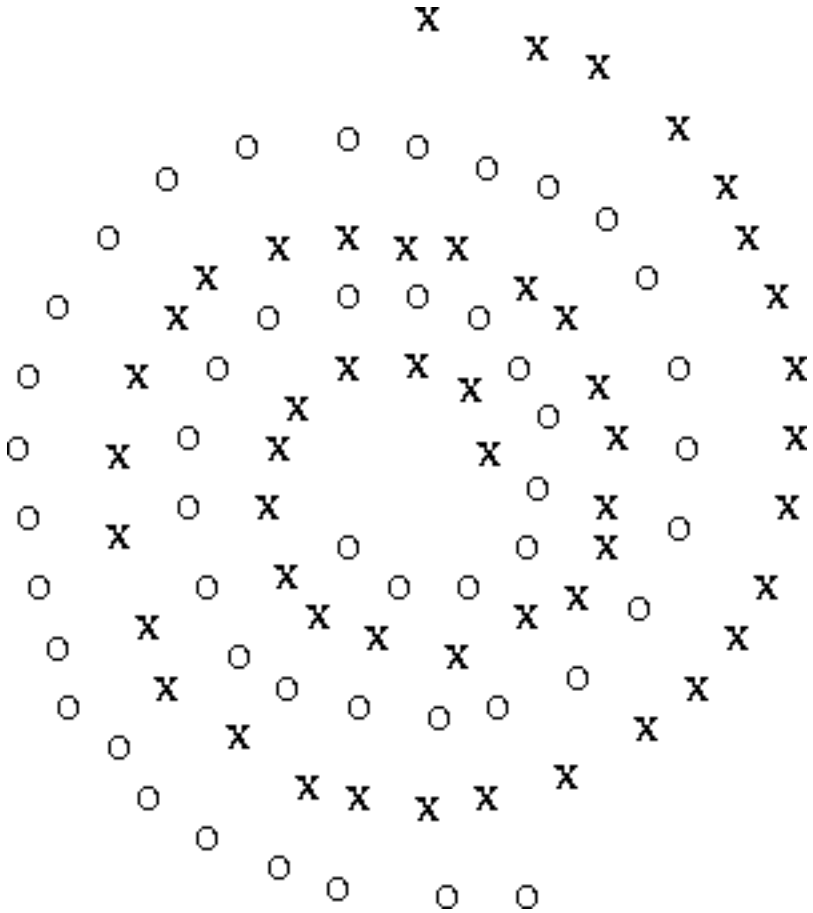
Next: Hopfield Networks **Up:** Other types of (Classical) **Previous:** The Kohonen Net

Probabilistic Neural Nets

Donald Specht published in the 1990 issue of *Neural Networks* mentioned in the bibliography a paper called *Probabilistic Neural Networks*. This followed earlier work which was not widely known to the Neural Net community which dates from the later sixties. The earlier work talked of polynomial discriminant functions and didn't mention neural nets which indeed were barely off the ground in those days.

One of the problems associated with MLPs, which is complained about from time to time by the more thoughtful users, is that whenever there is much structure in the data, the wildly hopping hyperplanes do not do a good job of collaborating to find it. A well known test case is the so called *double spiral* data set, shown in *Fig.* 5.19

Figure 5.19: The Double Spiral Data Set.



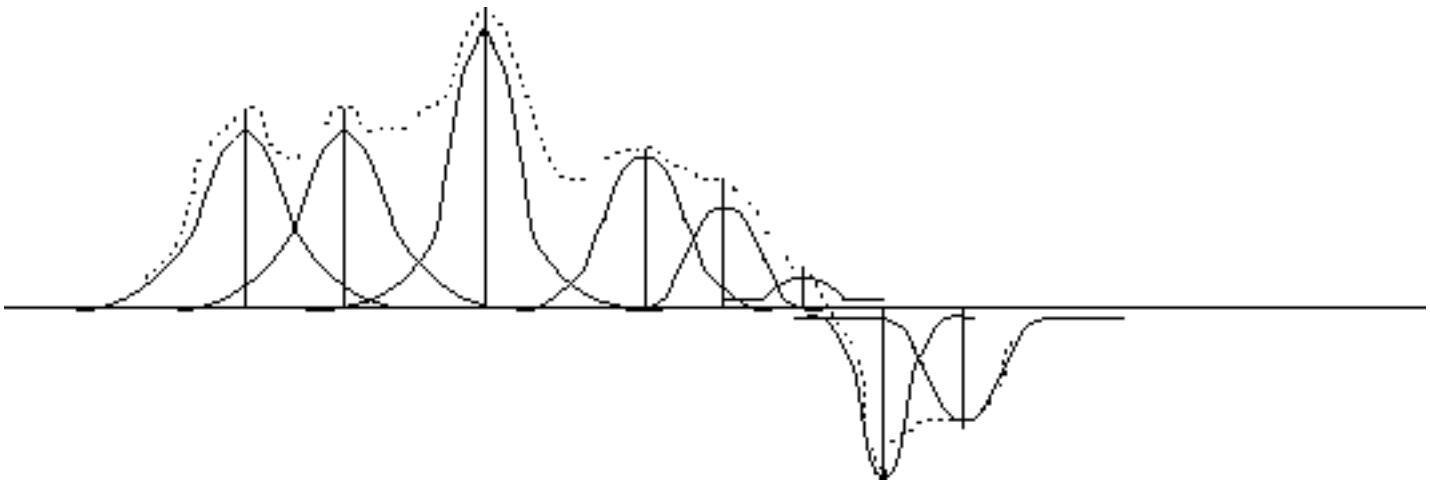
The prospect of training a four layer neural net to learn this arrangement of data is depressing to anyone who has ever tried, and also to the even smaller group who have thought about what it entails.

The difficulty of course lies in the restrictions of thought which Neural Netters have imposed upon themselves. Some of these restrictions arise from a training in mathematics which has emphasised the formal at the expense of the intuitive; some from an inability to abstract the essentials of a problem. What a Multi-layer Percepron does is to chop up the space into chunks, each chunk of which contains points from the data set of the same class. There is no compelling reason why the chunks should have boundaries which are parts of hyperplanes unless there is compelling reason to think that real neurons do in fact implement such sets, and there isn't. So we can choose any set which is convenient. Specht elects, in effect, to put spheres around the data points, and to label the spheres with the class of the data. He then takes the radius of the spheres as a parameter to be played with, and pumps it up or down until it interferes with spheres of a different class. He has the same radius for every sphere; he can interpret the sphere as one standard deviation of a spherical gaussian distribution and claim that he has a sort of gaussian mixture model for the data, constrained so as to have a covariance matrix which is some constant times the identity matrix. This allows him to not only classify the data, but also to have a probabilistic assessment of the category of another point. All the algorithm actually does is to take each data point as it comes in and centre a sphere on it. Then there is some messing about to decide what the radii of the spheres ought to be. By interpreting the sphere as one standard deviation of a gaussian distribution, it is possible to compute likelihoods for subsequent data. One usually avoids thinking of the geometry by describing the process by means of a covariance matrix which is some constant times the identity matrix, but only a small amount of thought shows that this defines a sphere, with the constant specifying the radius. It is easy to visualise the algorithm applied to the double spiral problem, and this allows one to see its merits and drawbacks rather clearly.

Parzen had earlier (1962) suggested essentially the same thing with a variety of different functions replacing the spherical gaussian.

The function fitting aspects of this procedure are worth developing; if instead of a finite number of categories each data point is assigned a real number, we can take a gaussian function centred on some point and multiply it by a number just sufficient to make the value of the scaled gaussian equal to the value assigned to the point. If we do this for every point in the data set and also pump up the standard deviation of the gaussians, we may get a reasonable approximation to a smooth curve or function through the data. *Fig.* 5.20 shows this done for the case where the data is in $\mathbb{R}$; the original values are given by spikes of the right height. We are, of course, under no compulsion to use gaussians, and function fitting by a generalised version of this is well known under the title of *radial basis functions*.

Figure 5.20: Radial Basis Functions, a.k.a. PNNs fitting a function



Putting a gaussian at each point of the data is rather wasteful, and this leads to trying to adapt the centres of the spheres so as to be somewhere in the middle of clusters of data and to have rather fewer of them. This is called using Adaptive Probabilistic Neural Nets, which in tune with modern methods of causing confusion are abbreviated to APNN. I shall say something about how to accomplish this later.

It may be asked what Adaptive Probabilistic Neural Nets have to offer over gaussian mixture modelling and the answer appears to be largely as given in the preamble to this chapter. There are packages one can use, or algorithms one can copy without the least understanding of what is actually going on. Function fitting or modelling probability density functions sounds, perhaps, less glamorous than supervised and unsupervised learning by Neural Nets. This is silly and indicates the naivety of much work in the area. Perhaps the reason that many writers in the area are obscure is because it conceals the poverty of the ideas. Where there are implementation issues, such as making PNN out of hardware or the (remote) possibility that they model real wetware neurons, these strictures of a professional grouch may be lifted somewhat, but one cannot help wishing that Mathematicians had done a better job of exposition of the ideas of their subject when teaching Engineers.

Probabilistic Neural Nets are not attractive from the point of view of their Rissanen complexity, since you get to send the whole data set plus at least one number specifying the radii of the gaussians. Adaptive PNNs are more attractive, and there are justifications of them which I shall consider later in the section on Quadratic Neural Nets, which contain APNNs as a special case. The network diagrams for PNN's do not convey much information except that it is possible to implement such things via parallel algorithms, which may or may not be an advantage. If the data looks like the double spiral data, one might wish that some acknowledgement of relationships between the bits were possible. But we shall return to such issues when we come to Syntactic Pattern Recognition.

Next Up Previous Contents

Next: Hopfield Networks **Up:** Other types of (Classical) **Previous:** The Kohonen Net *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Introduction **Up:** Other types of (Classical) **Previous:** Probabilistic Neural Nets

Hopfield Networks

The following sections have been written by Dr. Chris deSilva; we thought it wise that he should do this part because he is much more restrained and polite than I am, thus keeping down the costs of actions for libel.

- Introduction
- Network Characteristics
- Network Operation
- The Network Equations
- Theory of the Network
- Applications

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Network Characteristics **Up:** Hopfield Networks **Previous:** Hopfield Networks

Introduction

Hopfield networks were introduced in the early 1980's by John Hopfield, a well-known physicist. They originated as neurophysiologically-motivated models for **content-addressable memories** (CAM).

Content-addressable memories differ from conventional computer memories in that information is retrieved from a CAM by specifying either a part of the desired information or some properties of it, instead of specifying the location in the memory where the information is stored. This is reminiscent of the way in which human memory works; for example, we can recall a long word beginning with the letter A meaning *capable of using both hands with equal facility*.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Network Operation **Up:** Hopfield Networks **Previous:** Introduction

Network Characteristics

The Hopfield network works with input patterns which are vectors of real numbers. In his original paper [1], the input patterns consisted of binary vectors: each component was either or 1. We will use +1 and -1 instead; this simplifies the algebra. In a subsequent paper [2], Hopfield extended this to the case where the input patterns are vectors of numbers from some closed interval [*a*,*b*].

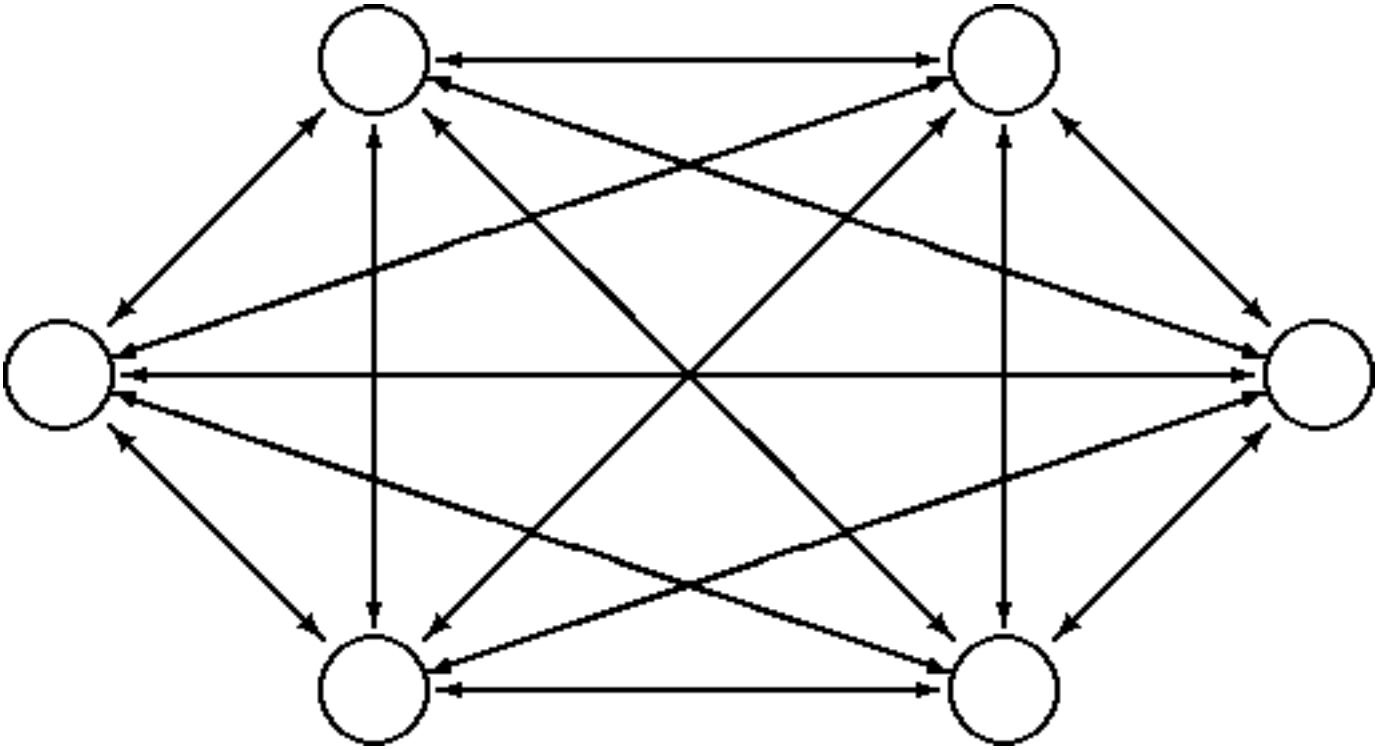
In all cases, the units of a Hopfield network have states, which are described by numbers belonging to the set of possible pattern values. In the binary case the states are either or 1, or +1 and -1. In the continuous case, the states are members of [*a*,*b*]. Patterns are input to the network by setting the states of the units to the appropriate values, according to some mapping from the components of the input vector to the units.

The Hopfield network is not trained in the way a Backpropagation network is. Instead, it resembles the Probabilistic Neural Network of Specht in that a set of exemplar patterns are chosen and used to initialize the weights of the network. Once this is done, any pattern can be presented to the network, which will respond by displaying the exemplar pattern that is in some sense similar to the input pattern. The output pattern can be read off from the network by reading the states of the units in the order determined by the mapping of the components of the input vector to the units.

The topology of Hopfield network differs from those of the two networks mentioned in the previous paragraph. There are no distinct layers; every unit is connected to every other unit. In addition, the connections are *bidirectional* (information flows in both directions along them), and *symmetric:* there is a weight assigned to each connection which is applied to data moving in either direction.

The figure shows a Hopfield network with six units.

Figure 5.21: A simple Hopfield network



Next Up Previous Contents

Next: Network Operation **Up:** Hopfield Networks **Previous:** Introduction *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: The Network Equations **Up:** Hopfield Networks **Previous:** Network Characteristics

Network Operation

As indicated above, the initialization of the Hopfield network stores a number of patterns by encoding them in the network weights. An input pattern is fed to the network by setting the states of the units to the levels determined by that pattern. The units are then allowed to interact as described below until the interaction reaches some steady state. This steady state will be one of the exemplar patterns, expressed in terms of the states of the units. (It is possible that the network will not attain a steady state, but will cycle through a sequence of states. We will not consider this case in the discussion to follow.)

The interaction between units is governed by the weights on the network connections. The weights must be set to values that will ensure that the eventual steady states are the ones representing the exemplar patterns. How these values are determined will be described below.

The interaction between the units can be thought of as a sort of competition between the units in which some units excite others, making them more active (that is, increasing the numbers describing their states), and some units inhibit others, making them less active (decreasing the numbers describing their states). The choice of weights ensures that these changes do not continue indefinitely.

An alternative way of considering the network is provided by a very simple model. Consider a billiard table whose surface is not flat, but slopes towards the pockets. The surface of the table can be taken to represent the possible states of a network with two units, and the pockets the positions of exemplar patterns. Supplying the network with an input pattern corresponds to putting a ball down on the table. The interaction between units until a steady state is reached is like the ball rolling down the sloping surface of the table and falling into a pocket.

Next Up Previous Contents

Next: The Network Equations **Up:** Hopfield Networks **Previous:** Network Characteristics *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Theory of the Network **Up:** Hopfield Networks **Previous:** Network Operation

The Network Equations

In general, let the network have *N* units, numbered from 1 to *N*. The weight assigned to the connection from the *i*th unit to the *j*th unit is w_{ij} for $1 \leq i, j \leq N$. Since no unit is connected to itself, $w_{ii} = 0$ for all $i = 1, \ldots, N$. Since the connection from the *i*th unit to the *j*th unit has the same weight as the connection from the *j*th unit to the *i*th unit, $w_{ij} = w_{ji}$.

Each unit has a threshold associated with it. The threshold determines the change in the state of the unit in response to inputs from the other units. We will denote the threshold of the *i*th unit by θ_i.

Suppose we have *M* exemplar patterns, $\mathbf{p}_1, \ldots, \mathbf{p}_M$, each of which has *N* components, so $\mathbf{p}_i = (p_i^1, \ldots, p_i^N)$ for $i = 1, \ldots, M$. We compute the weights according to the formula $w_{ij} = \sum_{k=1}^{M} p_k^i p_k^j$ for $1 \leq i, j \leq N, i \neq j$.

Let the state of the *i*th unit at time *t* be $\mu_i(t), i = 1, \ldots, N$. Suppose we have an input pattern $(x^1, \ldots, x^N)$ that is to be classified. We impose the pattern on the network by setting $\mu_i(0) = x^i, i = 1, \ldots, N$.

To run the network, we compute the states of the units at successive instants using the formula

$$\mu_i(t+1) = f\left(\sum_{j=1, j\neq i}^{N} w_{ij}\mu_j(t) - \theta_i\right)$$

for $i = 1, \ldots, N$ and $t = 1, 2, \ldots$

In the case where the units have binary outputs, *f* is the step funtion: $f(x) = -1$ if $x < 0$ and $f(x) = 1$ if $x > 0$.

(If $\sum_{j=1,j\neq i}^{N} w_{ij}\mu_j(t) - \theta_i = 0$ the state of the unit is not changed.) In the continuous case, *f* is a sigmoid function with range [*a*,*b*].

Next Up Previous Contents

Next: Theory of the Network **Up:** Hopfield Networks **Previous:** Network Operation *Mike Alder 9/19/1997*

Theory of the Network

The theory of the network is an extension and formalization of the billiard table model described above. Instead of the surface of the billiard table, we consider an energy surface over the state space of the network.

The state of a network with *N* units at time *t* can be described by a vector $(\mu_1(t), \ldots, \mu_N(t))$, where $\mu_i(t)$ is the output of the *i*th unit at time *t*. We define the **energy** at time *t* to be

$$E(t) = -\frac{1}{2}\sum_i \sum_{j \neq i} w_{ij}\mu_i(t)\mu_j(t) + \sum_i \mu_i(t)\theta_i$$

where θ_i is the threshold of the *i*th unit.

The energy of the network is analogous to the potential energy of a physical system. In the billiard table model, it corresponds to the difference between the height of the surface of the table at a point and the height of the nearest pocket. The choice of the energy function makes the Hopfield network essentially the same as a Ising model for spin glasses, which makes it of great interest to physicists.

The operation of the Hopfield network may be summed up as energy minimization. The choice of the network weights ensures that minima of the energy function occur at (or near) points representing exemplar patterns. The formula used to update the states ensures that the energy of each state is no greater than the energy of the preceeding state. The network rolls down the energy surface to the nearest exemplar pattern.

There are two methods for updating the states of the units. In the *sequential* method, the states of the units are changed one at a time, in random order. In the *simultaneous* method, the states of the units are all changed at the same time, on the basis of their states after the previous update.

It is easy to show that the sequential updating method decreases the energy at each step. Suppose that at time *t*, the state of the *k*th unit is changed. If the change in the state of the *i*th unit is $\Delta\mu_i(t+1) = \mu_i(t+1) - \mu_i(t)$, then we have $\Delta\mu_i(t+1) = 0$ for $i \neq k$, and

$\Delta\mu_k(t+1) \neq 0$.

The change in the energy caused by the change in the state of the *k*th unit is

$$\Delta E = -\frac{1}{2}\sum_i \sum_j w_{ij}(\mu_i(t+1)\mu_j(t+1) - \mu_i(t)\mu_j(t)) + \sum_i \theta_i(\mu_i(t+1) - \mu_i(t)).$$

Adding and subtracting $\mu_i(t+1)\mu_j(t)$, and simplifying, we get

$$\Delta E = -\frac{1}{2}\sum_i \sum_j w_{ij}(\mu_i(t+1)\Delta\mu_j(t+1) + \Delta\mu_i(t+1)\mu_j(t)) + \sum_i \theta_i \Delta\mu_i(t+1),$$

which can be reduced to

$$\Delta E = -\Delta\mu_k \left(\sum_i w_{ik}\mu_i - \theta_k\right),$$

using the facts that $\Delta\mu_i(t+1) = 0$ for $i \neq k$ and $w_{ij} = w_{ji}$.

The term in brackets is the same as the argument of the threshold function of the *k*th unit, so if its value is positive, the value of $\Delta\mu_k(t+1)$ is also positive, so ΔE is negative. If the value of the argument of the threshold function is negative, so is $\Delta\mu_k(t+1)$, and again ΔE is negative. Hence each change of state brought about by the sequential updating method reduces the value of the energy function.

The simultaneous updating method also leads to a reduction in the energy at each stage, but this is more difficult to show.

Since the operation of the network ensures that its energy is reduced at each update, it follows that the network will tend towards states that are local minima of the energy function. The positions of these minima are determined by the weights w_{ij}. The formula used to determine the weights from the exemplar patterns ensures that local minima of the energy function will occur at positions corresponding to the exemplar patterns, provided the number of exemplars is small in comparison to the number of units. It is usually stated that the number of patterns that can be effectively stored in the network is of the order of 15 per cent of the number of units. This makes the Hopfield network relatively inefficient as a memory device.

Next Up Previous Contents

Next: Applications **Up:** Hopfield Networks **Previous:** The Network Equations *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: The Boltzmann Machine **Up:** Hopfield Networks **Previous:** Theory of the Network

Applications

The principal application of the Hopfield network has been by physicists to the theory of spin glasses. There have been uses of the network for associative or content-addressable memory, in spite of its inefficiency. The architecture has also been the inspiration for networks which minimise functions other than the energy function introduced by Hopfield.

An application of Hopfield networks to the problem of recognizing handwritten digits is described in [3], as part of a comparison of the performance of a number of network architectures on this problem.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Introduction **Up:** Other types of (Classical) **Previous:** Applications

The Boltzmann Machine

- Introduction
- Simulated Annealing
- Network Characteristics
- Network Operation
- Theory of the Network
- Applications

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Simulated Annealing **Up:** The Boltzmann Machine **Previous:** The Boltzmann Machine

Introduction

The energy function of the Hopfield network is determined by the exemplar patterns. The network weights are fixed by the patterns and there is no means of having the network learn the weights through exposure to them. The Boltzmann machine extends the capabilities of the Hopfield network by introducing an algorithm for the adaptive determination of the weights.

The Hopfield network finds local minima of the energy function. In many cases, this is sufficient, but in many cases, too, it is desirable to find the state which is the global minimum of the energy function. The Boltzmann machine combines the Hopfield network architecture with the process called **simulated annealing** in an effort to find a global minimum of the energy function.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Network Characteristics **Up:** The Boltzmann Machine **Previous:** Introduction

Simulated Annealing

Annealing is a process used in glass making and metallurgy. It is the process of solidifying glass or metal from the molten state by cooling it slowly. Slow cooling results in a far more orderly arrangement of atoms and molecules than rapid cooling, with the result that the resulting solid is far stronger.

In minimization problems, a basic obstacle to the determination of the global minimum is the presence of local minima. Iterative methods for seeking a minimum, such as hill climbing, can get trapped in a neighbourhood of a local minimum, because escaping from such a neighbourhood would involve a temporary increase in the value of the function that is to be minimized.

The solution to this problem proposed by simulated annealing is to suspend the minimization process occasionally, in the hope that this will allow the process to escape from a local minimum to some other minimum, and possibly to the global minimum. The process is controlled by a parameter which is varied during the course of the process is such a way as to ensure that escape may occur frequently in the early stages of the process, and less and less frequently as the process continues. This parameter is the analogue of the temperature in physical annealing, where the temperature controls the mobility of atoms and molecues, making them less mobile as the substance cools.

Simulated annealing is an iterative minimization process in which the system moves from state to state, subject to the temperature parameter. At each stage, the system can move either to a state for which the energy is higher than its present state, or to a state of lower energy. The decision as to which state is made randomly, the probabilities of each being determined by the temperature parameter. Early in the process the temperature parameter is high, as is the probability of moving to a state of higher energy. As the process continues, the temperature parameter is reduced, and with it the probability of moving to a state of higher energy. This procedure is applicable to a number of different minimization procedures, and has been found to be effective in finding the global minimum or minima close to the global one.

Next Up Previous Contents

Next: Network Characteristics **Up:** The Boltzmann Machine **Previous:** Introduction *Mike Alder 9/19/1997*

Network Characteristics

There are a number of different descriptions of the Boltzmann machine in the literature. The following is based on the paper of Ackley *et al.* [4].

The Boltzmann machine is a fully connected network with symmetric bidirectional connections. In addition, certain units of the network are designated as *input units*, certain other units are designated as *output units*, while the remainder are the *hidden units*. This division of the network is entirely arbitrary, since the units are indistinguishable.

The units in the network can be in one of two states, which may be denoted as *on* and *off*, and 1, or 1 and -1. The network has an energy function of the same form as that of the Hopfield network:

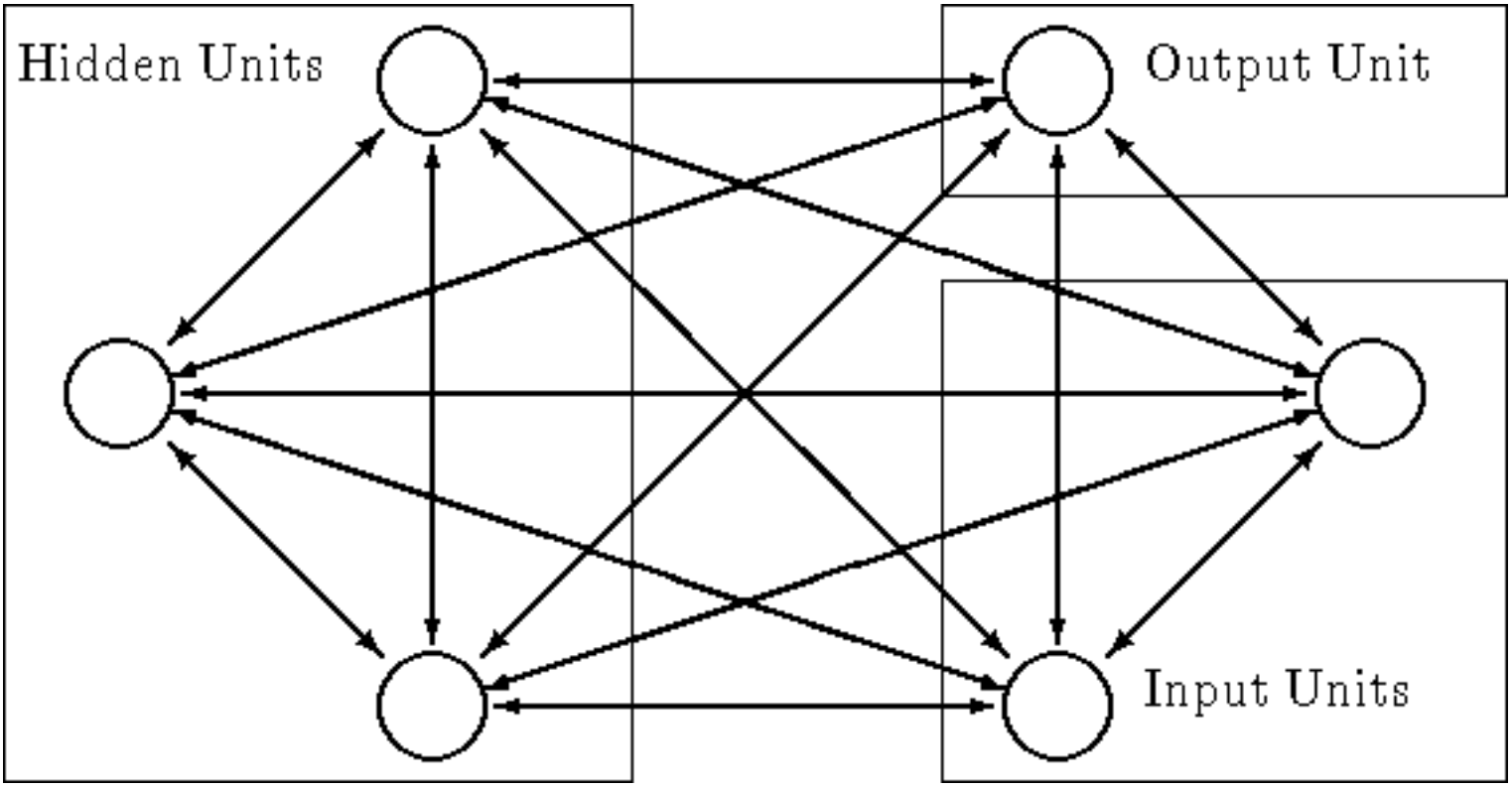
$$E(t) = -\frac{1}{2}\sum_i \sum_{j \neq i} w_{ij}\mu_i(t)\mu_j(t) + \sum_i \mu_i(t)\theta_i$$

where $\mu_i(t)$ is the state of the ith unit at time t, w_{ij} is the weight on the link between the ith and jth units, and θ_i is the threshold of the ith unit.

The network also has a control parameter T, the temperature parameter, which controls the simulated annealing process when the network is run.

The figure shows a simple Boltzmann machine.

Figure 5.22: A simple Boltzmann machine



Next Up Previous Contents

Next: Network Operation **Up:** The Boltzmann Machine **Previous:** Simulated Annealing *Mike Alder 9/19/1997*

Network Operation

The determination of the network weights of the Boltzmann machine is a two-step process. In the first step, the states of the input and output units are held fixed and the states of the hidden units are modified by the simulated annealing process until a steady state is achieved. When this happens, statistics on the frequency with which pairs of units are both *on* are collected. This process is repeated for all the exemplar pairs several times. In the second step, only the states of the input units are held fixed, and the states of the hidden units and the output units are modified by the simulated annealing process until a steady state is reached. When this happens, statistics on the frequency with which pairs of units are both *on* are again collected. The statistics collected are used to update the network weights and the procedure is repeated until the weights stabilise.

Suppose the network has N_i input units, denoted $\mu_1, \ldots, \mu_{N_i}$,N_h hidden units,

$\mu_{N_i+1}, \ldots, \mu_{N_i+N_h}$, and N_o output units, $\mu_{N_i+N_h+1}, \ldots, \mu_{N_i+N_h+N_o}$. Put $N = N_i + N_h + N_o$. The weights w_{ij} and thresholds $\theta_i, 1 \leq i, j \leq N$, are initialized to some arbitrary values,

subject to $w_{ij} = w_{ji}, i \neq j$,and $w_{ii} = 0$.

The following procedure is repeated until the weights stabilise.

The input and output pairs are selected in some order for presentation to the network. The states of the input and output units are fixed at the values determined by each selected input and output pair, the states of the hidden units are set to some random initial values, and the temperature is set to some high value.

Simulated annealing is used to bring the network to a steady state. The hidden units are chosen in some random order and the quantity $\Delta E = \sum_j w_{ij}\mu_j(t) - \theta_i$ is computed, where the chosen unit is

the ith one, and $\mu_j(t)$ is the state of the jth unit at time t. A random number between and 1, denoted ρ,

is also chosen. The state of the chosen unit is udpated according to the following rules:

- if $\Delta E < 0$ the state of the unit is changed;
- if $\Delta E \geq 0$ and $\rho < \exp(-\Delta E/T)$ the state of the unit is changed;
- otherwise the state of the unit is not changed.

After cycling through the units a number of times, the temperature parameter is reduced and the

procedure is repeated. When the annealing schedule is completed, the network is run for a number of cycles while the states of pairs of units are examined, and the pairs which are both in the *on* state are recorded.

After all the input and output pairs have been presented a number of times, the recorded results from each presentation are used to compute p_{ij}, the proportion of the time that both the *i*th and *j*th units were *on*.

The exemplar patterns are now presented to the network again, but this time only the states of the input units are held fixed, while the states of both the hidden units and the output units are changed according to the simulated annealing process. Statistics are collected for the computation of p'_{ij}, the proportion of the time that both the *i*th and *j*th units were on when the output units were free.

The weights are now updated. If $p_{ij} - p'_{ij} > 0, w_{ij}$ is incremented by a fixed amount. If $p_{ij} - p'_{ij} < 0, w_{ij}$ is decremented by a fixed amount.

Once the weights stabilize, the network can be used for recall. The states of the input units are fixed at the values determined by the selected input vector and the simulated annealing procedure is applied to the hidden units and the output units. When equilibrium is reached, the output vector can be read off the output units.

Next Up Previous Contents

Next: Theory of the Network **Up:** The Boltzmann Machine **Previous:** Network Characteristics *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Applications **Up:** The Boltzmann Machine **Previous:** Network Operation

Theory of the Network

The theory of the network is based on statistical mechanical concepts. In particular, it can be shown that when the network is allowed to run according to the simulated annealing procedure, the states of the network occur with a fequency that depends on the Boltzmann distribution. This distribution is a fundamental one in statistical mechanics. The distribution is determined by a single parameter, which is the temperature in thermodynamical situations, and has the property that states with low energy are unlikely to occur when the temperature parameter is large, and much more likely to occur when the temperature parameter is small.

The Boltzmann distribution provides the rationale for the simulated annealing procedure. Starting the procedure at high temperature makes the network more likely to occupy high energy states. As the temperature is reduced, the low energy states become more likely. Very low temperatures favour very low energy states, which should be close to the global minimum.

The Boltzmann distribution also provides the rationale for the procedure used to update the weights. It can be shown that the quantity $p_{ij} - p'_{ij}$, described above, is an indicator of the direction and amount of the change in w_{ij} that will result in a better matching of the input and output patterns.

Next Up Previous Contents

Next: Applications **Up:** The Boltzmann Machine **Previous:** Network Operation *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Bidirectional Associative Memory **Up:** The Boltzmann Machine **Previous:** Theory of the Network

Applications

Ackley *et al.* [4] applied the Boltzmann machine to the **Encoder Problem**, which involves an attempt to create an efficient description of the input vectors by encoding them in the states of the hidden units. It also can be applied to the study of the dynamics of spin glasses.

The Boltzmann machine was among the architectures applied to the recognition of handwritten digits in [3].

An application of the Boltzmann machine to job-shop scheduling is presented in [5] and [6].

The principal limitation the Boltzmann machine is the length of time it takes to produce its results, both in training the weights and in recall. In addition, in spite of the use of simulated annealing, it can get stuck in local minima and fail to find the global minimum.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Introduction **Up:** Other types of (Classical) **Previous:** Applications

Bidirectional Associative Memory

- Introduction
- Network Characteristics
- Network Operation
- The Network Equations
- Theory of the Network
- Applications

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Network Characteristics **Up:** Bidirectional Associative Memory **Previous:** Bidirectional Associative Memory

Introduction

Bidirectional Associative Memory (BAM) is an extension of the Hopfield network introduced by Kosko [7]. The principal difference between BAM and the Hopfield network is that the units are divided into two sets and there are no connections between the units in each set. The BAM is described as being capable of learning pattern pairs, but this is a misleading description of its operation; it simply recalls stored patterns, which have been previously been divided into two parts by the network designer.

Mike Alder
9/19/1997

Network Characteristics

The BAM is not a fully connected network. Instead, the units are divided into two sets, or layers. The layers are generally designated as the *A* and *B* layers; the number of units in the *A* layer need not be the same as the number of units in the *B* layer. Each unit in each layer is connected to all the units in the other layer and to none of the units in the layer to which it belongs. The links are bidirectional.

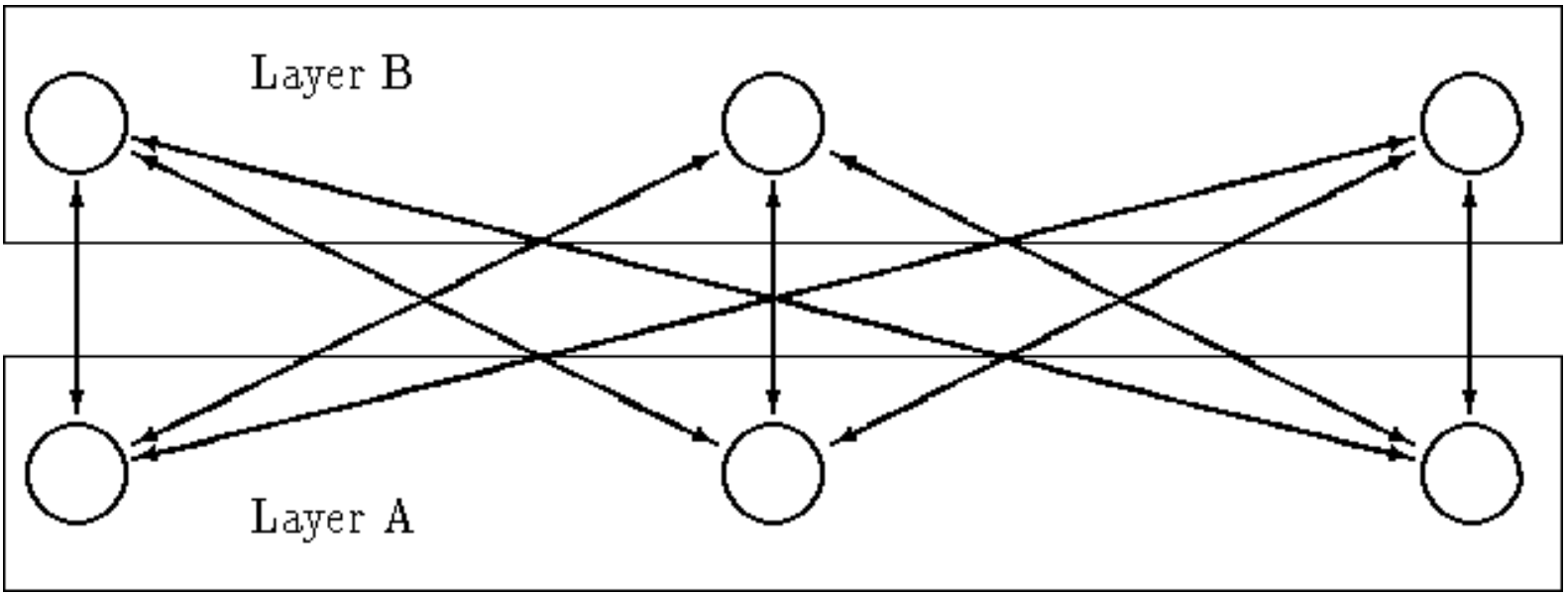
The units in the network can be in one of two states, which may be denoted as *on* and *off*, and 1, or 1 and -1. If the network has N_A units in the *A* layer and N_B units in the *B* layer, we will denote the state of the *i*th unit in the *A* layer at time *t* by $\mu_i(t)$ and the state of the *j*th unit in the *B* layer at time *t* by $\nu_j(t)$. The weight on the link joining the *i*th unit in the *A* layer to the *j*th unit in the *B* layer is w_{ij}, where $1 \leq i \leq N_A$ and $1 \leq j \leq N_B$.It may not be the case that $w_{ij}=w_{ji}$; in fact, since N_A need not be the same as N_B, in some cases both w_{ij} and w_{ji} may not be defined.

The network has an energy function whose form is similar to that of the Hopfield network:

$$E(t) = -\sum_{i=1}^{N_A}\sum_{j=1}^{N_B} w_{ij}\mu_i(t)\nu_j(t).$$

The figure shows a simple BAM.

Figure 5.23: A simple Bidirectional Associative Memory



Next Up Previous Contents

Next: Network Operation **Up:** Bidirectional Associative Memory **Previous:** Introduction *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: The Network Equations **Up:** Bidirectional Associative Memory **Previous:** Network Characteristics

Network Operation

As in the case of the Hopfield network, the weights are determined from a set of exemplar pattern pairs. The weights are computed once.

To run the network, the states of the units in the *A* layer are set according to some chosen pattern. The states of the units in the *B* layer are then determined for the states of the units in the *A* layer and the weights. New states for the units in the *A* layer are then computed from the states of the units in the *B* layer and the weights. This cycle between the layers is repeated until a steady state is reached.

Mike Alder
9/19/1997

The Network Equations

The weights are determined from a set of exemplar pattern pairs. If there are *P* pairs, $(\mathbf{a}_1, \mathbf{b}_1), (\mathbf{a}_2, \mathbf{b}_2), \ldots, (\mathbf{a}_P, \mathbf{b}_P)$,where $\mathbf{a}_k = (a_k^1, \ldots, a_k^{N_A})$ and

$\mathbf{b}_i = (b_k^1, \ldots, b_k^{N_B})$ and $a_k^i = \pm 1, b_k^j = \pm 1$ for

$1 \leq i \leq N_A, 1 \leq j \leq N_B, 1 \leq k \leq P$, then the weights are given by

$w_{ij} = \sum_{k=1}^{P} a_k^i b_k^j, 1 \leq i \leq N_A, 1 \leq j \leq N_B$.

Let the initial states of the units in the *A* layer be set to $\mu_i(0)$ for $1 \leq i \leq N_A$. At time $t > 0$ we compute new states for the units in the *B* layer by computing the activations $\beta_j(t) = \sum_{i=1}^{N_A} w_{ij}\mu_i(t-1)$ and then setting the new states of the units to $\nu_j(t) = 1$ if

$\beta_j(t) > 0, \nu_j(t) = \nu_j(t-1)$ if $\beta_j(t) = 0$ and

$\nu_j(t) = -1$ if $\beta_j(t) < 0, 1 \leq j \leq N_B$.

We then compute new states for the units in the *A* layer by computing the activations $\alpha_i(t) = \sum_{j=1}^{N_B} w_{ij}\nu_j(t)$ and then setting the new states of the units to $\mu_i(t) = 1$ if

$\alpha_i(t) > 0, \mu_i(t) = \mu_i(t-1)$ if $\alpha_i(t) = 0$,and $\mu_i(t) = -1$ if $\alpha_i(t) < 0$,

$1 \leq i \leq N_A$.

New states for the units in each layer are computed in alternation until a steady state is reached, which represents the output of the network.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Applications **Up:** Bidirectional Associative Memory **Previous:** The Network Equations

Theory of the Network

The theory of the network is similar to that of the Hopfield network. It can be shown that each cycle reduces the value of the energy function, so that the network tends to a local minimum of the energy function. The determination of the weights ensures that the exemplar patterns are located at these minima, provided the number of patterns is small compared to N_A and N_B.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: ART **Up:** Bidirectional Associative Memory **Previous:** Theory of the Network

Applications

The poor storage capacity of the BAM architecture has limited its applicability.

There have been extensions of the architecture to cases where the states are continuously variable and where the weights are adapted over time instead of being fixed by the set of exemplar patterns.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Introduction **Up:** Other types of (Classical) **Previous:** Applications

ART

- Introduction
- Network Characteristics
- Network Operation
- Theory of the Network
- Applications

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Network Characteristics **Up:** ART **Previous:** ART

Introduction

Adaptive Resonance Theory (ART) was developed by Stephen Grossberg in his studies of the behaviour of models of systems of neurons. Grossberg has published a large number of papers, in which he develops the mathematical theory of such models, usually in terms of systems of differential equations, and applies this to actual neural systems. Like Kohonen, he has been particularly interested in systems that are capable of organizing themselves.

Grossberg has used his mathematical results to develop two neural network architectures, which implement nearest neighbour pattern classification. The first architecture, usually referred to as **ART1**, works on patterns of binary data. The second, **ART2**, is an extension of ART1 to arbitrary patterns. ART1 is described in [8], ART2 in [9] and [10].

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Network Operation **Up:** ART **Previous:** Introduction

Network Characteristics

Both the ART architectures are two-layer networks in which units in the first, or input, layer receive inputs of the data and feedback from the second layer. Units in the second, or output, layer receive inputs from the input layer and interact among themselves.

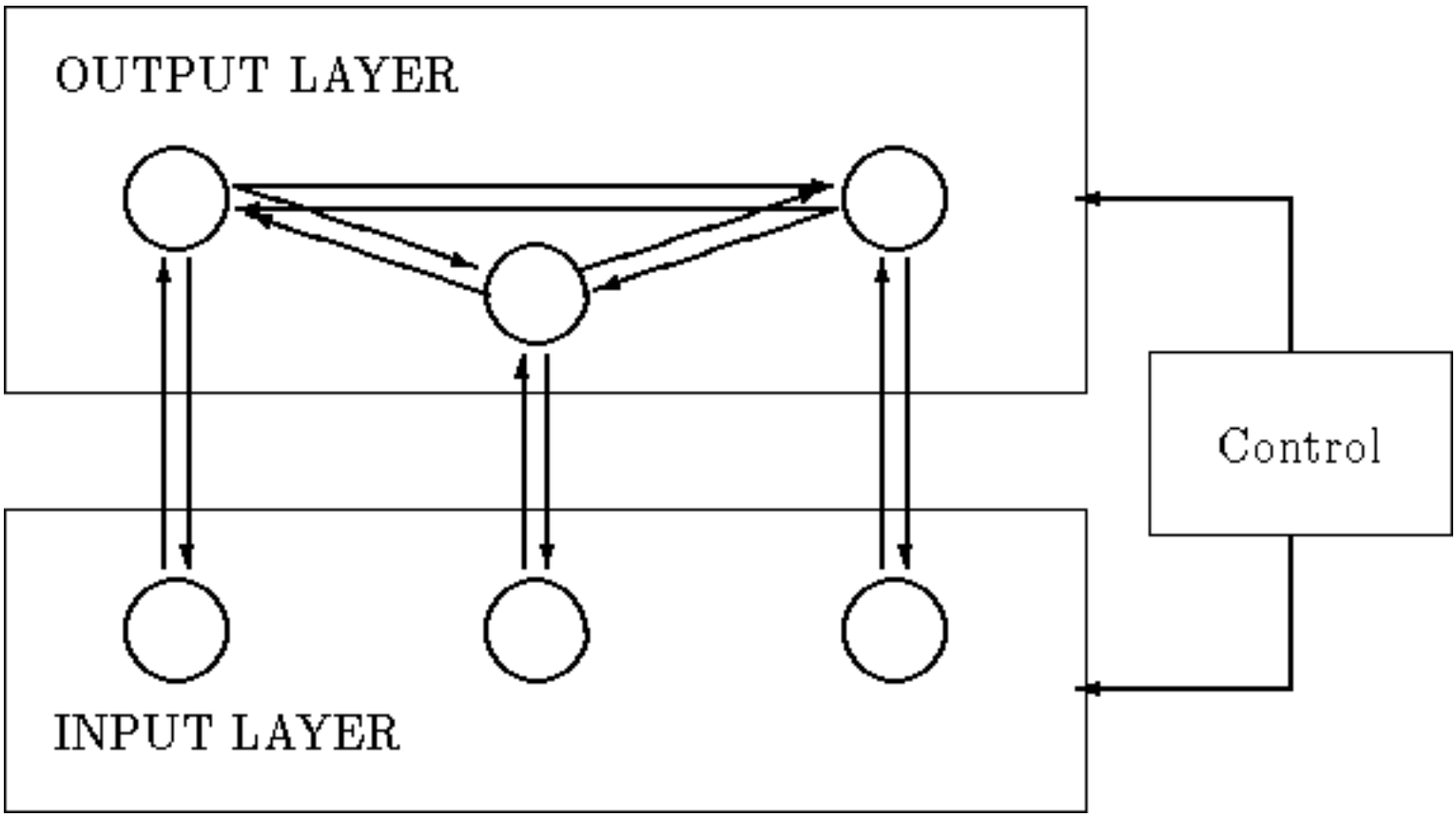
The ART networks implement a nearest neighbour classification algorithm whose simplicity is obscured by the complexity of the network architecture.

The network stores patterns as sets of weights assigned to the paths connecting the input units to each of the output units. Presentation of an input vector causes the output units to be activated, the amount of activation depending on the similarity between the input vector and the stored pattern. Finding the nearest neighbour is simply a matter of deciding which output unit has been activated the most.

The networks also have a mechanism for adding new units to the output layer. This mechanism is invoked whenever an input is presented which is dissimilar to all the existing stored patterns. The invocation of this mechanism is regulated by a control parameter. In fact, the control of the operation of the network is a complex process requiring interactions between the units in both layers and a control unit.

The figure shows a simple ART1 network.

Figure 5.24: A simple ART1 network



Next Up Previous Contents

Next: Network Operation **Up:** ART **Previous:** Introduction *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Theory of the Network **Up:** ART **Previous:** Network Characteristics

Network Operation

The heart of the ART networks is a scheme for interaction between the units in the output layer which guarantees that the most active unit will suppress the activity of all the other units. After the output units have been activated, they are allowed to interact until only one remains active. This unit then activates the units in the first layer via another set of paths with their own weights. The input vector is then compared with the pattern that has been passed down from the output layer.

The comparison between these patterns is controlled by a parameter called the *vigilance threshold*, which determines the degree of similarity between patterns that are given the same classification. If the difference between the input pattern and the pattern that is passed down from the output layer is less than the vigilance threshold, then the active output unit is accepted as determining the correct classification of the input, and the weights associated with that unit are adjusted according to one of the schemes described below.

If the difference between the input pattern and the pattern passed down from the output layer is greater than the vigilance threshold, the activity of the active unit is completely suppressed, and process described above is repeated until either a classification is achieved, or there are no units left, in which case a new class is created by the addition of a unit to the output layer.

The weights that connect each output unit to the input units represent the pattern typical of the class to which the output unit belongs. The weights that connect the input units to each output unit represent the same pattern, except that their values are normalized.

The effect of the operation is as follows: for each input vector, the network finds the closest pattern among the output units. If this is within the distance determined by the vigilance threshold, this is accepted as the correct classification and the weights adjusted to make the stored pattern more similar to the input vector. Otherwise, the network continues to go through the remaining output units in order of similarity to the input vector. If any of them is found to be within the distance determined by the vigilance threshold, it is chosen as the correct classification. If no output pattern is sufficiently similar, a new output unit is created.

Both ART1 and ART2 operate in this way. The only difference between the two is that ART1 inputs consist of binary patterns, while the inputs to ART2 networks can have arbitrary values, which are normalized by additional operations that are applied within the input units.

Next Up Previous Contents

Next: Theory of the Network **Up:** ART **Previous:** Network Characteristics *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Applications **Up:** ART **Previous:** Network Operation

Theory of the Network

The equations that are used to determine the states of the units are based on the results of Grossberg's research into systems of differential equations. The principal theoretical result underpinning the operation of the ART networks is the *Cohen-Grossberg Stability Theorem.* This is a very general and technical result about a class of ordinary differential equations. In the context of ART, it guarantees that the units in the output layer will reach a steady state and not oscillate periodically or chaotically.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Neocognitron **Up:** ART **Previous:** Theory of the Network

Applications

Grossberg and his collaborators have applied the ART architectures to many problems, including modelling of human behaviours [11] and classification of radar images [12].

The Defense Science and Technology Organization in Adelaide has applied ART1 to the classification of infra-red ship images [13].

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Introduction **Up:** Other types of (Classical) **Previous:** Applications

Neocognitron

- Introduction
- Network Structure
- The Network Equations
- Training the Network
- Applications

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Network Structure **Up:** Neocognitron **Previous:** Neocognitron

Introduction

The Neocognitron was developed by Fukushima [14] in an attempt to construct a neural network architecture based explicitly on knowledge about real brains. It is probably the most complex neural network architecture developed to date, and probably the most limited in terms of the scope of its possible applications.

The Neocognitron was designed to recognize handwritten digits. Much of its complexity stems from an attempt to make the recognition process robust against variations in different people's handwriting and against variations in the position of the digit presented to it.

The design of the Neocognitron was inspired by knowledge of the visual system derived from neurophysiological experiments such as those of Hubel and Wiesel ([15] and [16]). These experiments have discovered a great deal about the structure and function of the parts of the brain that derive their inputs primarily from the retina. In particular, it has been discovered that there is only a small number of types of cells receiving inputs directly from the retina and these cells have very restricted functions; that the local relationships between cells in the retina are preserved in the organization of the neural pathways; and that the visual system appears to be structured hierarchically, with information being passed from one level of the structure to the next. These discoveries are reflected in the design of the Neocognitron.

There are a number of features that make the Neocognitron different from other neural network architectures. These include its *hierarchical structure*, its *training methodology*, and its *network dynamics*. These features will be described in greater detail below.

The hierarchical structure of the Neocognitron is perhaps the most significant difference between it and other network architectures. It makes it possible to piece together the global picture in a number of steps, so that the amount of processing to be done by any one layer is limited. In addition, each of the units receives input from only a small fraction of the units in the previous layer, reducing the number of connections between units.

Next Up Previous Contents

Next: Network Structure **Up:** Neocognitron **Previous:** Neocognitron *Mike Alder*
9/19/1997

Network Structure

The Neocognitron is designed as a system to recognize the handwritten digits from 0 to 9. It has a hierarchical structure, being divided into nine layers.

The first layer is the input layer and is designated U_0. It consists of an array of photoreceptor units arranged in a 19×19 square. The remaining layers are divided into four sets of two layers each.

Each of these sets has a U_S layer and a U_C layer, which are given numbers to indicate the set to which they belong. The flow of information is along the path

$$U_0 \rightarrow U_{S1} \rightarrow U_{C1} \rightarrow U_{S2} \rightarrow U_{C2} \rightarrow U_{S3} \rightarrow U_{C3} \rightarrow U_{S4} \rightarrow U_{C4}.$$

The U_{S1} layer has $19 \times 19 \times 12$ units. Each unit in the U_0 layer has 12 units corresponding to it in the U_{S1} layer. Each of these units receives inputs from the corresponding units in the U_0 layer and from the eight units surrounding it. Each of these twelve units is trained to respond to a pattern that represents a small portion of a line segment. In this way, the $S1$ units function as a collection of line segment detectors spread over the entire input array.

The U_{C1} layer consists of $11 \times 11 \times 8$ units. The 12 bit patterns recognized by the $S1$ units actually correspond to 8 possible orientations of line segments, so outputs from equivalent $S1$ units are fed to the same $C1$ units. Each $C1$ unit receives inputs from $S1$ units corresponding to a 5×5 square of receptor units. This reduces the dimensions of the layers from $19 \times 19 \times 12$ to $11 \times 11 \times 8$.

There are similar arrangements between the succeeding layers, so that the U_{S2} layer has $11 \times 11 \times 38$ units, the U_{C2} layer has $7 \times 7 \times 22$ units, the U_{S3} layer has $7 \times 7 \times 32$,the U_{C3} layer has $7 \times 7 \times 30$, the U_{S4} layer has $3 \times 3 \times 16$, and the U_{C4} layer has ten units, which are used to indicate which of the digits has been presented to the network. The gradual reduction in the first two dimensions of each layer reflects the gradual increase in the proportion of the input array which affects each unit. The alternating reduction and increase in the size of the third dimension reflects encoding of a number of features in the output from the previous layer and the subsequent reduction of this number by identifying features that are equivalent for the purposes of digit recognition.

The units described above are all excitatory units. In addition, there are cells between each of the U_C layers and the following U_S layer which have an inhibitory effect on the units in the following layer. A

little arithmetic shows that there are more than 50000 units in the Neocognitron.

The Neocognitron is a feedforward network with the outputs from each layer (except the last) becoming the inputs to the next layer. It is not a fully connected network; there are no connections between units belonging to the same layer, and each unit is connected to some, rather than all, of the units in the previous layer. This means that the number of connections is less than that of a Backpropagation network of similar size and far less than that of a comparable Hopfield network. Even so, the Neocognitron has about 14 million connections.

Next Up Previous Contents

Next: The Network Equations **Up:** Neocognitron **Previous:** Introduction *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Training the Network **Up:** Neocognitron **Previous:** Network Structure

The Network Equations

The output of each unit at any time is represented by a non-negative real number. The connections between units have weights associated with them which modify the inputs from the previous layer.

The equations describing the operation of the units are complex, and the description which follows is condensed and simplified.

Each unit in each of the U_S layers takes the outputs from the excitatory units in the previous layer to which it is connected, multiplies them by the appropriate weights, and sums the products. We will denote this sum by *e*. It also has an input from an inhibitory unit, which is multiplied by its own weight. This product is denoted by *h*. The effects of the excitation and inhibition are combined by computing $x=((1+e)/(1+h))-1$. The output of the unit is 0 if $x<0$ and *x* otherwise. The output will be equal to *e* if $h=0$ and 0 if $h>e$.

Each unit in each of the U_C layers takes the outputs from the excitatory units in the previous layer to which it is connected, multiplies them by the appropriate weights, and sums the products. We will denote this sum by *x*. The output of the unit is $x/(\alpha_l + x)$ is $x \geq 0$ and 0 otherwise. α_l is a parameter associated with the *l*th layer which determines the degree of saturation of the output of all the units in that layer.

The inter-layer inhibitory units each send an output to a single unit in the following U_S unit and receive inputs from the same units in the preceeding U_C layer as the cell which receives its output. The output of each inhibitory unit is the weighted root mean square of its inputs.

Next Up Previous Contents

Next: Training the Network **Up:** Neocognitron **Previous:** Network Structure *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Applications **Up:** Neocognitron **Previous:** The Network Equations

Training the Network

The network is trained one layer at a time. For each layer, a set of training patterns is defined, and the desired responses of the units in the layer to those patterns is determined. The patterns are presented one at a time, and the weights on the connections to the units in the layer are adjusted until the units respond as required.

The training procedure differs from that of other networks in two ways. First, as mentioned above, the network is trained a layer at a time. The training is supervised, and directed towards having the units in each layer behaving in a predetermined way to each input pattern. This requires that the behaviour of all the units in all the layers be specified in advance, and that the network can only work effectively on the class of patterns on which it was trained. Training for the digits will require a different procedure from training for ten alphabetic characters.

Second, the training is replicated across the input array. This is most easily explained in relation to the U_{S1} layer. There are 12 units in this layer that correspond to each unit in the input (U_0) layer. The intention of the training process is that the twelve units at each position respond to the same set of input features. To ensure this, changes to the weights are determined for a single position in the input array, say at (x_0,y_0), and applied to the weights at all positions in the next layer, namely at (x,y,z_0), for all x and y. This method ensures that the network is insensitive to variations in the positions of the input patterns.

Once the network is trained, presentation of an input pattern should result in one of the units in the U_{C4} layer having the largest output value. This unit indicates which of the digits was presented as input.

Next Up Previous Contents

Next: Applications **Up:** Neocognitron **Previous:** The Network Equations *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: References **Up:** Neocognitron **Previous:** Training the Network

Applications

The Neocognitron was designed as a system for recognizing handwritten digits. It may be used for other pattern recognition tasks by training it on an appropriate data set. However, the network is so complex and requires so much specialized training and tuning that it is unlikely to have many applications.

Mike Alder
9/19/1997

References

The following papers were cited above.

[1] Hopfield, J.J. Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences*, **79**, pp. 2554-2558, 1982.

[2] Hopfield, J.J. Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of the National Academy of Sciences*, **81**, pp. 3088-3092, 1984.

[3] Pawlicki, T.F., Lee, D.-S., Hull, J.J., and Sihari, S.N. Neural network models and their application to handwritten digit recognition, *Proceedings of the IEEE International Conference on Neural Networks*, vol. II, 1988, pp. 63-70.

[4] Ackley, D.H., Hinton, G.E., and Sejnowski, T.J. A learning algorithm for Boltzmann machines, *Cognitive Science*, **9**, pp. 147-169, 1985.

[5] Foo, Y.-P. S., and Takefuji, Y. Stochastic Neural Networks for Solving Job-Shop Scheduling: Part 1. Problem Representation, *Proceedings of the IEEE International Conference on Neural Networks*, vol. II, 1988, pp. 275-282.

[6] Foo, Y.-P. S., and Takefuji, Y. Stochastic Neural Networks for Solving Job-Shop Scheduling: Part 2. Architecture and Simulations, *Proceedings of the IEEE International Conference on Neural Networks*, vol. II, 1988, pp. 283-290.

[7] Kosko, B. Bidirectional associative memories, *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-18**, pp. 42-60, 1988.

[8] Carpenter, G. and Grossberg, S. A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics and Image Understanding*, **37**, pp. 54-115, 1987.

[9] Carpenter, G. and Grossberg, S. ART2: Self-organization of stable category recognition codes for analog input patterns, *Proceedings of the IEEE First International Conference on Neural Networks*, vol. II, 1987, pp. 727-736.

[10] Carpenter, G. and Grossberg, S. ART2: Self-organization of stable category recognition codes for analog input patterns, *Applied Optics*, **26**, pp. 4919-4930, 1987.

[11] Grossberg, S. and Mingolla, E. Neural dynamics of perceptual grouping: Textures, boundaries and emergent segmentations, *Perception and Psychophysics*, **38**, pp. 141-171, 1985.

[12] Carpenter, G. and Grossberg, S. Invariant pattern recognition and recall by an attentive self-organizing ART architecture in a nonstationary world, *Proceedings of the IEEE First International Conference on Neural Networks*, vol. II, 1987, pp. 737-746.

[13] Lozo, P., Johnson, R.P., Nandagopal, D., Nussey, G., and Zyweck, T. An Adaptive resonance Theory (ART) based Neural Network Approach to Classifying Targets in Infrared Images, *Proceedings of the Second Australian Conference on Neural Networks*, 1991, pp. 22-25.

[14] Fukushima, K., Miyake, S., and Ito, T. Neocognitron: a neural network model for a mechanism of visual pattern recognition, *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-13**, pp. 826-834, 1983.

[15] Hubel, D.H., and Wiesel, T.N. Receptive fields, binocular interaction and functional architecture in cat's visual cortex, *Journal of Physiology*, **160**, pp. 106-154, 1962.

[16] Hubel, D.H., and Wiesel, T.N. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat, *Journal of Neurophysiology*, **28**, pp. 229-289, 1965.

The following books were also used in preparing these sections.

Anderson, J. A., and Rosenfield, E. *Neurocomputing: Foundations of Research*, The MIT Press (Cambridge, Mass.), 1989.

Beale,R., and Jackson, T. *Neural Computing: An Introduction*, Adam Hilger (Bristol), 1990.

Hecht-Nielsen, R. *Neurocomputing*, Addison-Wesley (Reading, Mass.), 1991.

Simpson, P.K. *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations*, Pergamon Press (New York), 1990.

Next Up Previous Contents

Next: Quadratic Neural Nets: issues **Up:** Other types of (Classical) **Previous:** Applications *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Summary of Chapter Five **Up:** Other types of (Classical) **Previous:** References

Quadratic Neural Nets: issues

The MLP cuts up the space of a finite binary data set into pieces by hyperplanes. This allows us to fit functions from a family so as to get agreement on the function sample defined by the data set, and the algorithmic process consists of shuffling hyperplanes about a space until they cut the data up in a satisfactory way.

If you were to be given a two dimensional problem with data consisting of noughts and crosses or black circles and white circles and invited to place a collection of hyperplanes judiciously so as to separate the classes, you could probably do it faster than a computer could do it using a neural net. For some easy problems such as the double spiral data set you could do it in a few seconds, while it is likely to take mini-supercomputers a few weeks using Back-Propagation for three layer nets. Even given the remarkable powers of the eye-brain combination, this does not inspire much confidence in Back-Prop.

Some reflection on the reasons why Back-Prop is so bad, or a little sensible experimenting, lead you to see that you are hoping that a collection of hyperplanes will fall into a nice set of positions, each complementing the others. Why should they? There is nothing at all in Back-Prop to make them. Fahlman pointed this out (see the references), and suggested more thoughtful ways of going about the job of partitioning data than taking random walks in a huge state space. Whether the resulting algorithm can reasonably be called a neural net algorithm is open to dispute, but it sure out-performs Back-Prop.

A little more reflection suggests that doing it with hyperplanes is pretty dumb anyway. If you are going to do it using the smallest number of units, you are likely to want to decompose your sets of points into a union of convex sets, as in the case of the articulated four layer net. Why not start off with convex sets and plonk them onto the data? Specht did this for spheres, why not use ellipses in two dimensions or hyperellipsoids in general? They are all specified by a positive definite quadratic form, or alternatively by a symmetric positive definite matrix. That's no big deal, since defining a simplex, the simplest closed convex shape you can make out of hyperplanes, in dimension n takes n+1 hyperplanes, each of which takes n numbers. This is about twice the amount of information it takes to specify a quadratic form.

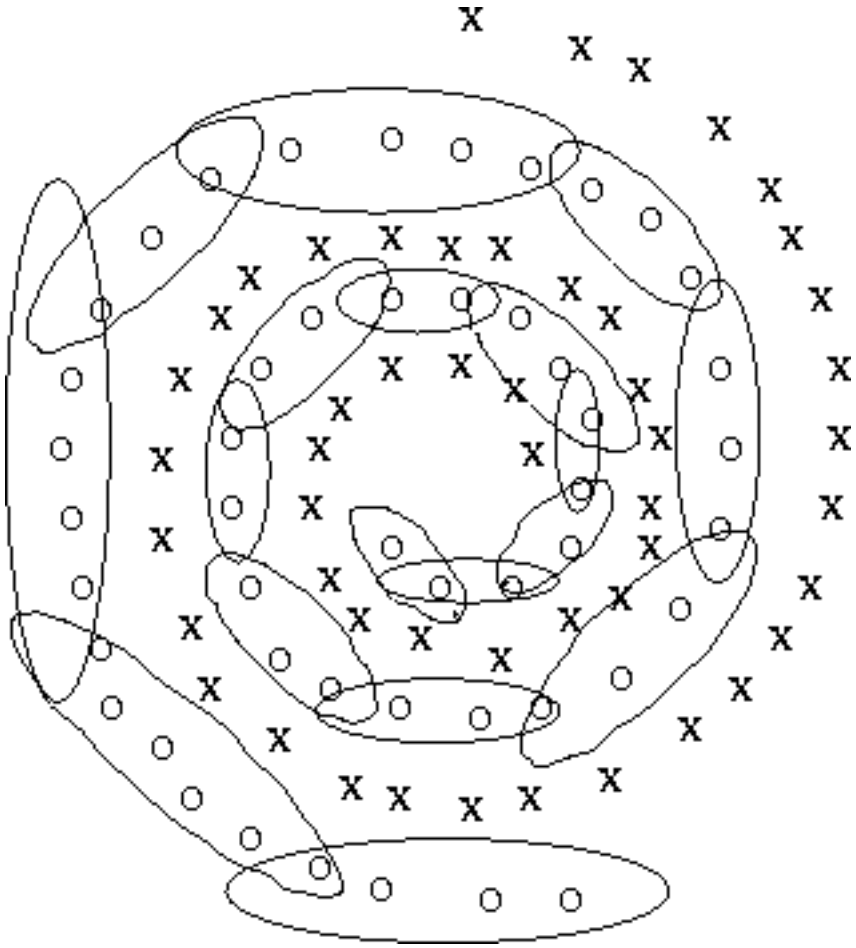
Which leads to a number of questions, one of which might be: what, if anything, does any of this have to do with neurons in people's heads? Are quadratic forms more plausible than affine discriminants?

I shall argue later that the answer is `yes'.

The next question is, If you want to tell noughts from crosses in the double spiral problem, for instance, how do ellipses help you? The answer is that you have two species of ellipses, red ellipses for the noughts and green ones for the crosses. Then you attract the red ellipses in towards the nought data, the green one toward the cross points, and they totally ignore each other. If your rule of attraction is judiciously chosen, you wind up with the answer to the next question, how do you train the quadratic neurons? How can I attract them towards the data? I'll go into that a little later on, too. It requires a little thought.

Fig. 5.25 shows a partial solution to the double spiral problem which may be obtained in times four or more orders of magnitude faster than may be found using the classical piecewise affine neural nets, that is to say it's at least ten thousand times faster. The artist got bored of drawing vile shaped ellipses and so only put them on the O's, leaving the reader to draw some on the X's if he feels so inclined, after which a small child may be invited to colour them in, yielding an image much like one of those described above.

Figure 5.25: A partial solution to the Double Spiral Data Set.



A case may be made out, and indeed will be made, for the proposition that real, live, wetware neurons do something rather like fitting quadratic or higher order forms to data. The effect of this may be something not too different from an adaptive Probabilistic Neural Net, and it tends to suggest that neurons are actually making local estimates of the density of data. This is an important conception of the function of a neuron, so deserves some extended exegesis. This will be done in a later chapter.

For the present, it suffices to observe that if data comes in through sensors in a stream, extracting information may require an adaptive process somewhat different from usual statistical methods for computing model parameters. Moreover, although a gaussian *pdf* is determined by a quadratic form, a quadratic form may be used to define other *pdf*s or may be used as are hyperplanes simply as delimiters of different regimes.

This way of looking at things unifies some otherwise disparate matters. One satisfying feature of Quadratic neural nets is that enclosing points of different categories by hyperellipsoids rather than by hyperplanes makes Neural Net methods look a whole lot more like statistics and gaussian mixture modelling, as well as being many thousands of times faster on many problems. A second is that classifying different types of point by having different categories of model neuron responding to them, reduces classifying to multiple clustering. So function fitting and clustering have at least some common elements from a neural net point of view.

After I've told you how to make positive definite quadratic forms, otherwise hyperellipsoids, jump through hoops and find data points, you will never want to use Back Propagation with Multi-layer Perceptrons for classifying data ever again. This will save you a lot of computer time.

Next Up Previous Contents

Next: Summary of Chapter Five **Up:** Other types of (Classical) **Previous:** References *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Exercises **Up:** Decisions: Neural Nets(Old Style) **Previous:** Quadratic Neural Nets: issues

Summary of Chapter Five

We started the chapter with a brief and possibly prejudiced account of some of the history of neural nets. We raised the issue of the extent to which they could accomplish wonders beyond the capacity of statistical methods, or whether connectionists should be having a mid-life crisis. We continued by first studying the alpha perceptron or single unit neural net, and proceeded by choosing to investigate the geometry of committee nets (three layered neural nets, in the conventional usage) and then of articulated nets with an extra layer. This allowed us to gain some insight into the principles of chopping up, by hyperplanes, the measurement space for a pattern classification task. We also constructed some simple algorithms which extended the perceptron convergence algorithm for a single unit to the case of these three layer and four layer nets. The Back-Propagation algorithm could then be seen in perspective: it was easy to see what it did and how it did it, and to perceive its practical limitations. The Back-Prop algorithm was outlined; the details of the algorithm were brushed off in rather an off-hand manner with a referral to the literature on the grounds that (a) clever people like you and me could now work it out from first principles if that looked like a good idea and (b) it looked a pretty bad idea. A discursion into function fitting followed, again with the intention of casting doubts on the standard procedures involving multi-layer perceptrons for doing this job, and shaking any ingenuousness the reader might have about the value of the contribution of pure mathematicians, who are as desperate to publish anything that might be mistaken for an article of practical value as any of us, and a good deal worse equipped to recognise practical value if they should chance to meet it.

The next section considered the value of Feed Forward MLP's from the point of view of their effectiveness as models from the Rissanen data compression perspective. I argued that an MLP accomplishing a binary classification was using hyperplanes to store only the category of the data and said nothing about where the data is in the space. It is thus easy to compute the compression effected by a neural net of this type and compare it with the null model to see if it is actually extracting information about the disposition of the points of a category. We concluded that the number of data points must be more than nkP in order for the model to have any value, where n is the dimension of a committee net, k the number of units and P the precision, some value between 2 and 64 which was in principle data dependent. The situation was worse for more complex net topologies. This suggested a certain element of frivolity in some artificial neural net publications.

A brief survey of neural nets other than MLP's was conducted, the thrust being to expose the underlying ideas rather than to specify the algorithms in any great detail. It was considered sufficient to describe what the things actually did and outline how they did it, because this is often enough to put one off the whole idea of getting involved with them.

Finally, quadratic neural nets were introduced in order to show that they did what some of the others did, and did it more intelligently and consequently much faster. The case was weakened by omitting details which, it was promised, would emerge later.

Next Up Previous Contents

Next: Exercises **Up:** Decisions: Neural Nets(Old Style) **Previous:** Quadratic Neural Nets: issues *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Bibliography **Up:** Decisions: Neural Nets(Old Style) **Previous:** Summary of Chapter Five

Exercises

1. Construct a binary data set in the plane. Colour the points of one category red and the other green, and display them on a computer screen. Construct a three layer neural net for a two dimensional input, and display the lines representing the weights of the units in the hidden layer; a small arrow sticking out of the side of each plane should be added to indicate which is the positive side. Lock all the last weights to the output unit at value 1. Program your machine to present data from the double spiral at random, use Back-Propagation to adapt the hidden unit weights, and display the (oriented) lines on the screen. Run it with varying numbers of hidden units and monitor the Square Error. Verify that the program is behaving itself by working with simple data sets first. Compare the times for convergence for data sets of varying complexity, and observe the behaviour of the net when there is no solution. Run your program on a double spiral data set.
2. With random initialisations of a set of gaussians, try solving the above data sets using the EM algorithm. Display the gaussians as ellipses using the methods of chapter four. Use red ellipses for the red data points, green ellipses for the green data points, and run them quite independently. Use a Bayesian method for determining the correct category; also use the Mahalanobis distance to the closest ellipse centre.
3. Increase the dimension progressively for both of the above systems. You will not be able to display the results graphically now, but keep records of times to convergence using a variety of random initialisations for both systems.
4. What conclusions do you draw from observing the performance of these two systems on different kinds of data set?

Next Up Previous Contents

Next: Bibliography **Up:** Decisions: Neural Nets(Old Style) **Previous:** Summary of Chapter Five *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Continuous Dynamic Patterns **Up:** Decisions: Neural Nets(Old Style) **Previous:** Exercises

Bibliography

1. G.F. Simmons, *Introduction to Topology and Modern Analysis*, McGraw-Hill, 1963.
2. Teuvo Kohonen, *Self-organization and associative memory, 3rd Edn.* Berlin ; New York : Springer-Verlag, 1989.
3. Michael Alder, Roberto Togneri, Edmund Lai and Yianni Attikiouzel, *Kohonen's algorithm for the numerical parametrisation of manifolds* Pattern Recognition Letters 11(1990)pp313-319.
4. Donald Specht, *Probabilistic Neural Networks* Neural Networks 1990
5. M.D. Alder, R. Togneri and Y. Attikiouzel *Dimension of the Speech Space* IEE Proceedings-I Vol 138, No. 3, June 1991.
6. Sok Gek Lim, Michael Alder & Paul Hadingham *Adaptive Quadratic Neural Nets* Pattern Recognition Letters,13, May 1992, pp325-329.
7. T. Khanna, *Foundations of Neural Networks* Addison Wesley 1990.
8. Jacek Zurada *Introduction to artificial neural systems* St. Paul: West, 1992 .
9. Michael Alder, Sok Gek Lim, Paul Hadingham and Yianni Attikiouzel *Improving Neural Net Convergence* Pattern Recognition Letters 13 (1992)pp331-338.
10. S. Fahlman and C. Lebiere *The Cascade-correlation learning architecture* in D.S. Toretzky (Ed) *Advances in Neural Information Processing Systems 2* Morgan Kauffman, 1990.
11. R. Togneri, M.D. Alder and Y. Attikiouzel *Dimension and Structure of the Speech Space* IEE Proceedings-I Vol.139, No.2, April 1992.

Mike Alder

9/19/1997

Next Up Previous Contents

Next: Automatic Speech Recognition **Up:** An Introduction to Pattern **Previous:** Bibliography

Continuous Dynamic Patterns

The first chapter of this book gave an overview from a considerable altitude, with all the lack of detail that compels. The second chapter got stuck into practical issues of measurement of images as a sort of antidote, since in real life much hangs on the details and they need to be treated squarely. The third chapter was so theoretical as to alarm many a simple soul, but was necessary for the thinking man or woman in order to prepare them for a healthy degree of doubt about what follows, and some later discussion on neural models. The fourth chapter simply gave some convenient recipes for actually doing the job of recognising standard (static) patterns by statistical methods, and the last chapter explained how the standard neural nets work and how they accomplish pattern classification, when they do.

In the present chapter we start to deal with the problem of dynamic patterns, of things that change in time. Examples are speech, on-line character recognition, and such things as telling two missiles or two people apart, using knowledge of the way they move. I shall defer such issues as looking at video-images of moving objects later, until we have addressed the issue of how to work out what objects are in an image. In the present chapter I shall, in proper generality, discuss the issue of classifying trajectories in which arise from sampling some continuous process. In the next chapter I shall discuss issues arising from trajectories which occur in a discrete space of symbols known as an alphabet.

I shall start by discussing a simple and direct approach to Automatic Speech Recognition, the method being a staple of practical single word recognisers. This practical discussion will be followed by a short survey of the issues involved in continuous speech recognition. This problem is too hard to do much with at the present time, but the reasons it is so hard are interesting and worth thinking about.

The issue of *noise* in dynamic pattern recognition requires special treatment, it is something to which any measurement system is liable, and needs to be confronted. A great deal of engineering literature is concerned with this. I shall go briefly into the issues of stochastic processes or random time series and filters. I shall approach this from the engineering end rather than the statisticians end, although both have made contributions.

Since filtering theory and Automatic Speech Recognition (ASR) are both huge subjects in their own right, and quite impossible to cover in this book, the treatment will be confined to a survey of the basic ideas and pointers to the literature. The diligent reader will then, be in a position to write simple word recognition programs, will be in a position to understand and use samples of such programs on the accompanying disk, and will begin to understand the profound difficulties of the area.

Finally, I shall outline alternative approaches to Speech Recognition and discuss related problems.

- Automatic Speech Recognition
  - Talking into a microphone
  - Traditional methods: VQ and HMM
    - The Baum-Welch and Viterbi Algorithms for Hidden Markov Models
  - Network Topology and Initialisation
  - Invariance
  - Other HMM applications
  - Connected and Continuous Speech
- Filters
  - Linear Systems
  - Moving Average Filters
  - Autoregressive Time Series
  - Linear Predictive Coding or ARMA modelling
  - Into $\mathbb{R}^n$
  - States
  - Wiener Filters
  - Adaptive Filters, Kalman Filters
- Fundamentals of dynamic patterns
- Exercises
- Bibliography

Next Up Previous Contents

Next: Talking into a microphone **Up:** Continuous Dynamic Patterns **Previous:** Continuous Dynamic Patterns

Automatic Speech Recognition

- Talking into a microphone
- Traditional methods: VQ and HMM
  - The Baum-Welch and Viterbi Algorithms for Hidden Markov Models
- Network Topology and Initialisation
- Invariance
- Other HMM applications
- Connected and Continuous Speech

Mike Alder
9/19/1997

Talking into a microphone

If you say a word, perhaps the name of a digit, for example `one', into a microphone, then it is straightforward to sample and digitise the resulting signal, and feed it into a computer as a longish sequence of numbers measuring the voltage generated by the microphone and your voice. Typically, a word may take one third to half a second to enunciate, and the signal is sampled

perhaps twenty thousand times a second, giving around seven thousand numbers. Each number will be quantised to perhaps 12 or 16 bit precision. Thus we may be looking at a data rate of around 30 to 40 kilobytes per second. This present paragraph, would, if spoken at a reasonable reading rate, occupy over two megabytes of disk space. If printed, it would occupy around a kilobyte. There is therefore a considerable amount of compression involved in ASR.

There are various methods of proceeding from this point, but the most fundamental and conceptually simplest is to take a Discrete Fourier Transform (DFT) of a short chunk of the signal, referred to as the part of the signal inside a *window*. The FFT or Fast Fourier Transform of Cooley and Tukey accomplishes this with admirable efficiency and is much used for these purposes. I have already discussed the ideas involved in taking a Fourier Transform: there are two ways of thinking about it which are of value. The first is to imagine that we have a sound and something like a harp, the strings of which can resonate to particular frequencies. For any sound whatever, each string of the harp will resonate to some extent, as it absorbs energy at the resonant frequency from the input sound. So we can represent the input sound by giving the amount of energy in each frequency which the harp extracts, the so-called energy spectrum. This explanation serves to keep the very young happy, but raises questions in the mind of those with a predisposition to thought: what if the sound changes this spectrum in time? If it sweeps through the frequency range fast enough, there may not be enough time to say it has *got* a frequency, for example. And what kind of frequency resolution is possible in principle? These considerations raise all sorts of questions of how filters work that need more thought than many writers consider worthwhile.

The second way to look at the DFT and FFT is to go back to the old Fourier Series, where we are simply expanding one function from $[-\pi, \pi]$ to $\mathbb{R}$ in terms of an orthogonal set of functions, the sine and

cosine functions. This is just linear algebra, although admittedly in infinite dimensions.

Now we have to reckon with two issues: one is that the signal is sampled discretely, so in fact we have not an algebraically expressed function but a finite set of values, and the other is that the time window on which we do our expansion is constantly changing as we slide it down the signal. What sense does it make to take out a chunk of a continuously changing signal, pretend it is periodic, analyse it on that assumption, and then slide the window down to a different signal and do the same again? Of course, these two ways of looking at the problem give equivalent results, principally some honest doubt as to whether this is the way to go. Alternative transforms such as the Wigner transform and various wavelet

families are available for those who want to follow this line of thought. They are beyond the scope of this book, but the reader who is of a reflective disposition will be ready for them when he meets them. I shall skip these interesting isues on the grounds that they are somewhat too technical for the present work, which is going to concern itself with more mundane matters, but the reader needs to know that there are problems about fixing up the window so that the FFT gives acceptable answers

reasonably free of artefacts of the analysis process. See the standard works on Speech Recognition and many, many issues of the IEEE transactions on Signal Processing, and the IEE Proceedings part I for relevant papers.

We take, then, some time interval, compute the FFT and then obtain the power spectrum of the wave form of the speech signal in that time interval of, perhaps, 32 msec. Then we slide the time interval, the window, down the signal, leaving some overlap in general, and repeat. We do this for the entire length of the signal, thus getting a sequence of perhaps ninety vectors, each vector in dimension perhaps 256, each of the 256 components being an estimate of the energy in some frequency interval between, say, 80 Hertz and ten KHz. The FFT follows a divide and conquer strategy, and when it divides it divides by two, so it works best when the number of frequency bands is a power of two, and 256 or 512 are common. Instead of dealing with the raw signal values in the window, we may first multiply them by a window shaping function such as a Hamming Window, which is there to accomodate the inevitable discontinuity of the function when it is regarded as periodic. Usually this just squashes the part of the signal near the edge of the window down to zero while leaving the bit in the middle essentially unchanged.

Practical problems arise from trying to sample a signal having one frequency with a sampling rate at another; this is called `aliasing' in the trade, and is most commonly detected when the waggon wheels on the Deadwood Stage go backwards, or a news program cameraman points his camera at somebody's computer terminal and gets that infuriating black band drifting across the screen and the flickering that makes the thing unwatchable. There is a risk that high frequencies in the speech signal will be sampled at a lower frequency and will manifest themselves as a sort of flicker. So it is usual to kill off all frequencies not being explicitly looked for, by passing the signal through a filter which will not pass very high or very low frequencies. Very high usually means more than half the sampling frequency, and very low means little more than the mains frequency .

The 256 numbers may usefully be `binned' into some smaller number of frequency bands, perhaps sixteen of them, also covering the acoustic frequency range . The frequency bands may have their centres selected judiciously in a more or less logarithmic division of the frequency range, their widths also adjusted accordingly, and the result referred to as a simulated filterbank of sixteen filters covering the audio spectrum. Alternatively, you could have a bank of 16 bandpass filters, each passing a different part of the audio spectrum, made up from hardware. This would be rather old fashioned of you, but it would be faster and produce smoother results. The hardware option would be more popular were it not for the tendency of hardare to evolve, or just as often devolve, in time.

The so called `Bark' scale, slightly different from logarithmic and popularly supposed to correspond more closely to perceptual differences, is used by the more sophisticated, and, since speech has been studied since Helmholtz, there is an extensive literature on these matters. Most of the literature, it must be confessed, appears to have had minimal effect on the quality of Speech Recognition Systems.

Either of these approaches turns the utterance into a longish sequence of vectors in $\mathbb{R}^{16}$ representing the time development of the utterance, or more productively as a *trajectory*, discretely sampled, in $\mathbb{R}^{16}$. Many repetitions of the same word by the same speaker might reasonably be expected to be described as trajectories which are fairly close together in $\mathbb{R}^{16}$. If I have a family of trajectories corresponding to one person saying `yes' and another family corresponding to the same person saying `no', then if I have an utterance of one of those words by the same speaker and wish to know which it is, then some comparison between the new trajectory and the two families I already have, should allow us to make some sort of decision as to which of the two words we think most likely to have been uttered.

Put in this form, we have opened up a variant of traditional pattern recognition which consists of distinguishing not between different categories of point in a space, but different categories of *trajectory* in the space. Everything has become time dependent; we deal with changing states.

Note that when I say `trajectory' I mean the entire map of the time (discretely sampled) into the space, and not just the set of points in the path. These are very different things. If you walk across the highway and your path intersects that of a bus, this may not be important as the bus may have long since gone by. On the other hand if your *trajectory* intersects that of the bus, then unless it does so when you are both at rest, you are most unlikely to come back to reading this book or, indeed, any other. I am prepared to reconstruct the time origin of any two utterances so that they both start from time zero, but one may travel through the same set of points twice as fast as another, and the trajectory information will record this, while the *image* of the two trajectories will be the same. A trajectory is a function of time, the path is the image of this function. If you want to think of the path as having clock ticks marked on it in order to specify the trajectory, that's alright with me. Now it might be the case, and many argue that in the case of speech it *is* the case, that it is the path and not the trajectory that matters. People seldom mean it in these terms, since the words /we/ and /you/ are almost the same path, but the direction is reversed. Still, if two trajectories differ only in the speed along the path, it can be argued that they must sound near enough the same. This may or may not be the case; attempts to compare two trajectories so as to allow for differences in rate which are not significant are commonly implemented in what is known as *Dynamic Time Warping* (DTW). I shall not describe this method, because even if it applies to speech, it may not hold for everything, and DTW is more commonly replaced by Hidden Markov Modelling these days.

On first inspection, one might argue that a trajectory in one space is simply a point in a function space. This is true, but not immediately helpful, as different trajectories may not be in the same space, as function spaces are traditionally defined. It is rather hard to put a sensible metric structure on the set of maps from any interval of the real numbers into $\mathbb{R}^n$ without any other considerations. So the abstract extension from points to trajectories needs some extra thought, which may depend upon the nature of the data.

It would be a mistake to think that binning the power spectrum into some number n of intervals is the only way of turning speech into a trajectory in $\mathbb{R}^n$, there are others involving so called cepstra or LPC coefficients which are believed in some quarters to be intrinsically superior.

Next: Traditional methods: VQ and **Up:** Automatic Speech Recognition **Previous:** Automatic Speech Recognition *Mike Alder*
9/19/1997

Traditional methods: VQ and HMM

It appears to be an article of faith with practitioners of Automatic Speech Recognition (ASR) that an utterance is a string of phonemes and a phoneme is a region of the space of speech sounds. This is not quite what a phonetician means by a *phoneme*, but phoneticians are people who know, roughly, how to tell each other how to hold their faces so as to make noises like foreigners talking, and are generally quite unable to say much about the speech space. The anthropologists who feel the clash of disparate cultures have nothing on a speech engineer going to a meeting of phoneticians or vice-versa. The more courageous find it exhilarating, the less, terrifying.

If we follow, temporarily and without prejudice, the ASR engineers, then we solve the problem of recognition of trajectories by first chopping up the space into regions, and labelling each region with a symbol. Ideally, the region of the space would correspond to what a phonetician would call a phoneme. In practice we may have the region rather arbitrary in position and shape. A trajectory, complete with clock ticks, then gets turned into a sequence of symbols, the labels of the region it is in at each clock tick. If we have a large number of utterances, we obtain from our trajectories, and way of chopping the space up into regions, a corresponding number of symbol strings. We then try to model the strings corresponding to a word such as `yes' by means of a *stochastic grammar*, the strings corresponding to the word `no' by a different stochastic grammar, and a new word has its probabilities computed by applying each grammar in turn. Then we use Bayesian arguments to get back from the probability of each grammar having generated the string, to the probability, given the string, of one grammar or another having been the source. A *stochastic grammar*, to anticipate the next chapter somewhat, is an algorithm which assigns a probability to a string of symbols to say how likely it is to occur in a sample from a particular language, and a (stochastic) *language* is a set of strings of symbols with a frequency attached to each of them. It is clear that a certain collection of assumptions about the nature of speech have been employed to come to this approach.

Chopping up the space into lumps is known in the trade as *Vector Quantisation*, which no doubt sounds a little classier put that way, and may be accomplished in a variety of ways. Some of these are so arbitrary and whimsical as to be embarrassing. One of the better ways is actually to take the speech data as just a big stack of points in $\mathbb{R}^{16}$ or whatever, and use a gaussian mixture model with some judiciously chosen number of gaussians. Then each gaussian may be used to compute the likelihood of a given piece of speech sound having been obtained from it, and the largest likelihood yields the name of that particular gaussian as the symbol. Alternatively we can use the Mahalanobis distance from each of the centres. This gives somewhat better results than the conventionally used LBG algorithm, a variant of *k-means clustering* which I shall now describe.

Suppose we have a set of points in $\mathbb{R}^n$ and we wish to chop them up into clusters. We first decide how many clusters we propose to have, say *k*, and choose ourselves *k* different kinds of points, call them centres. Now we throw the centres down at random in the space, or perhaps uniformly, and then partition the points so as to assign each point to the closest centre, thus constructing subclusters. Then we move the centres to the centroids of the subclusters, and recompute the assignment of points to subclusters, and the shifting of centres to centroids of subclusters, until the process stabilises. We may have to get rid of centres which coincide, in extreme cases. This may be said in algebra:

Given $D \subset \mathbb{R}^n$, *D* finite and $K = \{x \in \mathbb{Z} : 1 \leq x \leq k\}$, let $f_0 : K \longrightarrow \mathbb{R}^n$ be an initial assignment of centres; this might be uniform on the region containing *D* or random. Now let $P_m = p_1, p_2, \cdots p_k$, for *m*=0 in the first instance, be the partition of *D* which has

$$\mathbf{x} \in p_i \Leftrightarrow \forall j \neq i \, ||\mathbf{x} - f_m(i)|| \leq ||\mathbf{x} - f_m(j)||$$

Now obtain, in general, f_{m+1} from f_m by putting

$$\forall i \in K, f_{m+1}(i) = \frac{1}{|p_i|} \sum_{\mathbf{x} \in p_i} \mathbf{x}$$

and similarly obtain another partition P_{m+1}.

It turns out to be rather too dependent on the initial assignment of centres to leave the thoughtful reader altogether happy.

In The LBG (Linde, Buzo, Gray, q.v.) algorithm, the *k* means are obtained by a splitting process: first we go to the centroid of the whole data set *D*. Then we choose a random line and split the centre into two centres, each moved apart a little distance along the line. Then we partition the data set by assigning each point of it to the closest centre, just as before. Then we move to the centroids of the partitioned bit, as before. And we proceed to replicate the splitting process until we get bored or run out of points. This leads to *k* being always a power of two. The method again is dependent on the orientation of the lines used and often leads to bad clusters, which is to say clusters that don't look like natural clusters to the naked eye. If we generate the data by a gaussian mixture, we can get some exceedingly unappealing results for the final LBG assignment of points to clusters, given that we know how we in fact did it. In abstract terms, this method places more significance on the metric than is normally warranted. It is true that many metrics will agree about where the centroid of a set of points should be, but they usually have a depressing indifference to the distribution of the data. Fitting gaussian clusters and computing Mahalanobis distances or likelihoods actually gives somewhat better performance in recognition systems, which suggests that reality doesn't love the standard metrics all that much. We can think of the quadratic form as being a local estimate of the metric tensor for the generating process, or if you prefer, the data is trying to tell you something about what `close' means, and it would be advisable to listen to it.

As a Vector Quantisation (VQ) system, gaussian mixture modelling by fairly crude means is as quick and easy and more defensible than most. The simplest method is to replace the k-means approach by computing not just the mean of the subcluster assigned to the centre, but also its covariance matrix, and then to use *that* in order to compute a Mahalanobis distance when doing the next partitioning of the data points. We can think of this as taking a local estimate of the metric, or a second order approximation to the distribution of data points. You will note that this is a sort of quick and dirty version of the EM algorithm.

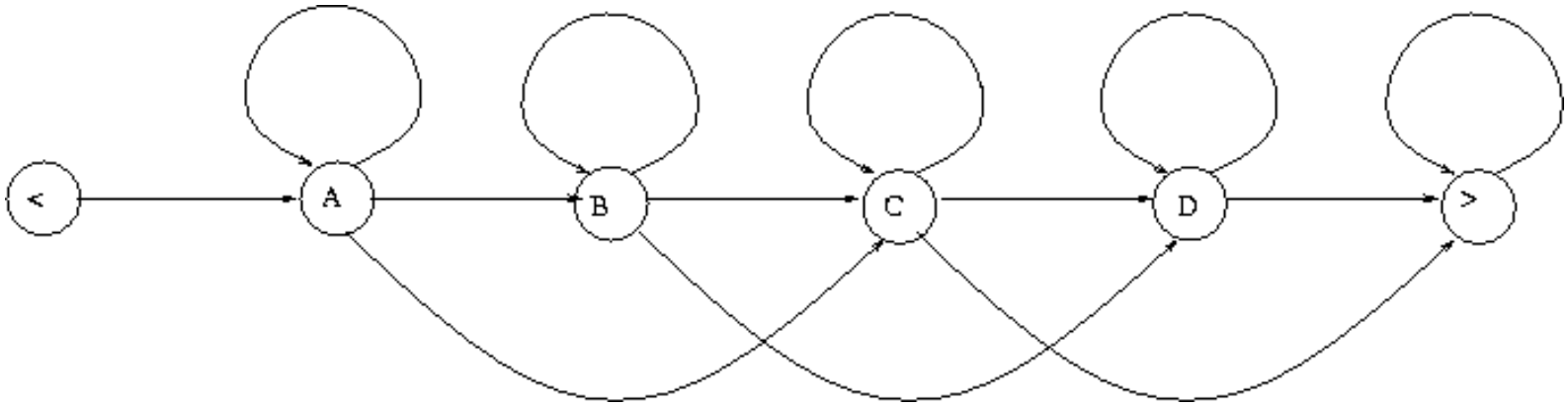
The labels for the different parts of the space are known as *codewords* and the list of them all is known as a *codebook*, for reasons which derive from coding theory. The general procedure of replacing points in a space by some set of central approximating points is used in many areas, for example in image compression, and it is the main idea in coding against noise, where the hope is that if you send the centres and they get corrupted, the points will be moved a short distance, but providing we choose to replace each point by the closest centre, we usually get back to where we started from. In this case a point is usually in a finite space of bytes.

It is common to try to estimate the extent to which replacing the actual point by the closest centre has deformed the trajectory, by adding up the squares of the distances of the actual points from the closest centres.This is called the *distortion measure* of the codebook. Of course, this presupposes that we know the right way to measure distances in the space, which we don't, so there is a certain fatuity in this approach. In image compression, for example, the Discrete Cosine Transform or DCT, a relative of the DFT which expands in terms of cosine functions only, is used on the two dimensional signal that is an image, and then the resulting function is quantised. Now the DCT produces a spectrum of spatial frequencies in the original image, and the eye is relatively insensitive to high spatial frequencies and low spatial frequencies, and is most sensitive somewhere in the middle. This is telling you that the euclidean metric on the space of DCT transforms is a particularly useless one to use if you want to say when two images look similar to human beings. Nor is there any reason to believe that the euclidean metric on a space of power spectra is of much use as a measure of acoustic similarity as perceived by people. So attempts to obtain distortion measurements by this method betray a certain naivete on the part of those making them. There's a lot of it about.

Having now chopped the space up into lumps and replaced the original trajectory by the sequence of codewords to which each point of the trajectory is closest (using, of course, a Mahalanobis distance in the case of the gaussian mixture modelling) we have the problem of representing the resulting strings by means of *stochastic grammars.* The stochastic grammars most commonly employed are known as *Hidden Markov Models* or HMMs for short. A hidden Markov model is

offered here, heuristically, as a description of the vocal tract going from one state to another. Typically ASR workers use a state diagram such as *Fig.* 6.1.

Figure 6.1: A Markov Process for spoken word recognition.



Here the states have been labelled for ease of talking about the diagram. Variants having more states or additional arcs (such as from A to D) are common.

We turn this diagram into a stochastic grammar for a set of strings, a language sample from a stochastic language, over the alphabet of labels of the regions into which the space has been chopped up into lumps, sorry, vector quantised. The stochastic grammar is obtained as follows.

Imagine someone in the next room to you stepping over the arcs represented by arrowed lines, between discs represented by the circles of *Fig.*6.1. Let them be supposed to have a handful of coins or dice or other means of generating random numbers. Suppose the diagram over which they trip their light fantastic, has been augmented in two ways. The first is that for each arc out of a node or disk, there has been a probability assigned. In going from node < to node A there is no choice, so we can put a probability of 1 on this arrow, but once at node A we have three choices, jumping up in the air and coming back down on node A, hopping to node B, or skipping to node C. In order to decide which to do, your colleague throws a die or dice, and makes a decision depending on the outcome. If the three probabilities (AA), (AB), (AC) were the same, then your friend might hop if the die showed one or two, skip if it showed three or four, and jump if it showed five or six. The probabilities out of each node all add up to one, and we shall suppose that every node with an arc leading out of it has had probabilities assigned to it.

The other augmentation is to suppose that for each arc there is a probability distribution over the alphabet which assigns a probability of your friend shouting out each symbol. Again, having chosen probabilistically an arc to move along, your friend now has, while moving along it, to make another throw of coins or dice in order to decide which symbol to call out. Tricky, but not impossible. You may recognise this as a first order markov process for the grammar.

As your friend next door starts from the < node and proceeds through the diagram, you hear the names of symbols being shouted out (along with the clatter of falling dice and coins), with a pause as she comes to the end node, >, and marches back to the beginning to do it again.

The system may be simplified, at modest cost, by ensuring that your friend calls out a symbol on arriving at a node, which means that the probability distributions over the symbols of the alphabet are the same for any two arcs terminating in the same node.

It would be easy to write a program to do the same, and less taxing for your friend.

The thing that is *hidden* about this markov model, is that you hear the string of symbols, but you don't know by which particular path through the network it was obtained. It may be possible that each string has a unique *parse*; that is to say, for any string there is either no way it could be obtained, or there is a unique sequence of hops, skips and jumps which was

responsible. On the other hand there might be a large number of different possible routes which could have yielded the string.

To set the model up by choosing probabilities assigned to arcs or symbols according to one's whim is easy enough but unprofitable. The question of interest is, suppose we have a diagram such as the last figure and a set of strings, how do we select the probabilities of arcs and symbols so as to get a good stochastic grammar for our sample? And other questions of interest are, where did that diagram come from, and is there a better choice? Avoiding the embarrassment of the last two questions *pro tem.*, let us suppose that we have obtained the diagram itself by inspecting the backs of the tablets on which the ten commandments were inscribed, but that the creator has neglected to insert the probability distributions (stored usually as tables and called the A-matrix and B-matrix in the literature). We may obtain some suitable numbers by the following procedure which the astute reader may recognise.

- The Baum-Welch and Viterbi Algorithms for Hidden Markov Models

---

Next Up Previous Contents

Next: The Baum-Welch and Viterbi **Up:** Automatic Speech Recognition **Previous:** Talking into a microphone *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Network Topology and Initialisation **Up:** Traditional methods: VQ and **Previous:** Traditional methods: VQ and

The Baum-Welch and Viterbi Algorithms for Hidden Markov Models

Start off with an arbitrarily chosen set of probabilities of choosing an arc from each node except the last, and another set of arbitarily chosen probabilities for each symbol being emitted on arriving at a node. Suppose you have to hand your sample of strings allegedly produced by this markov model. Now select the first string from the set of samples, and trace it through the diagram. If this cannot be done, some of your probabilities must have been made zero, which was naughty of you, go back and make them all positive. If it could be done in a lot of different ways, make a list of all of them. Since it is a finite length string and a finite diagram, there can only be a finite set of possibilities.

Each way of laying down the string from < to > through the diagram can have a probability computed for it, based on the arbitrarily assigned probabilities for transitions and emissions of symbols. The probabilities of choosing the sequences of arcs are readily computed by multiplying all the arc probabilities together, and the probability of having emitted a symbol at each arc, given that the arc was selected, can also be multiplied in at each symbol. So there is a net probability that the string was generated by any given parse, and we can add these together for all the different possible parses of the string. There are two approaches next, the *Viterbi* approach is to select the most probable parse and use that. The other approach due to Baum and Welch is to split up the string into fractional strings according to the probability of each parse, and lay down fractional strings over the diagram. If one imagines a diagram such as the above figure, and a string with symbols as beads on a piece of thread, the symbols have to get allocated to arcs or nodes in a sequence which lays out the thread over the diagram. In the case of the Baum-Welch method, we have lots of copies of the same threaded necklace of symbols, each with a weight attached to it, the sum of the weights being one. Each of them gets layed out on the diagram, making for lots of fractional threads everywhere in the Baum-Welch case, and just one in the Viterbi case. We maybe ought to make the threads we lay down on the diagram a nice striking colour, say scarlet, to contrast with the sober black of the network connections.

This is repeated for each string of the sample. This gives a diagram covered with a lot of scarlet threads. We then proceed to reestimate the probabilities using these threads. To reestimate the probability of travelling from A to C, we look at all the bits of string we have laid down, and count the total number leaving A. Of this total number (which for the Baum-Welch algorithm may not be an integer) we count the fraction going to C, and this fraction becomes the new probability assigned to the arc from A to C. Similarly for all the other arcs. Given that we go from A to C, we now count up the (possibly fractional) occurrences of each of the symbols that we have laid down along that path. The ratio of each symbol count to the total number of counts gives the probability reestimate for the symbol.

From the initial estimate and the sample of strings, we have a reestimate of the probabilities on the diagram. From the first, possibly ill chosen selection of probabilities, we have obtained a more plausible set which have done a better job of accounting for the actual data. We now do it again with the reestimate as our starting point. We repeat the process until the numbers don't change and claim that this gives a maximum likelihood model for the data given the network topology.

The Baum-Welch algorithm is clearly a particular case of the EM algorithm, which can be used for much more than just mixture modelling. The description given in English may be replaced either with some code or some algebra, but if you write your own code, making up a few little threaded necklaces of strings and working out where to put them will be much the quickest way. People of inferior intellects who have trouble with such arcana as reading and writing in natural languages may prefer algebra or C; such folk may find the above description confusing in which case they may find alternative formulations via the bibliography.

The process of effectively calculating all the probabilities may be organised, and the way of organising it is known as the *forward-backward algorithm.*

Next Up Previous Contents

Next: Network Topology and Initialisation **Up:** Traditional methods: VQ and **Previous:** Traditional methods: VQ and *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Invariance **Up:** Automatic Speech Recognition **Previous:** The Baum-Welch and Viterbi

Network Topology and Initialisation

There are, as anybody who has married, had children or been in politics is aware, some questions it is better not to ask. One of these is, where does the diagram of *Fig.* 6.1 come from? Another is, is a random initialisation of EM, or Baum-Welch, really a good idea? I shall ask them anyway.

If we suppose that the vocal tract has some fixed number of states and that any word consists of some definite number of them and we can decide which, but that the time spent in each state varies, and that sometimes states are skipped, then a diagram such as *Fig.* 6.1 becomes almost defensible against a weak and incoherent attack. If we take the view that it is a simple model which compresses strings and works, sort of, then the only useful attack upon it is to improve it, which has been done successfully.

It is found in the case of Hidden Markov Models, just as for Gaussian Mixture Modelling, that EM tends to be sensitive to initialisation, and for the former case, good initialisations for different words are passed between nervous looking individuals in seedy looking bars in exchange for used notes of small denomination. At least, nobody publishes them, which one might naively think people would. But this is based on the assumption that people publish papers and books in order to inform readers about useful techniques, and not to impress with one's supernal cleverness. To be fairer, initialisations are word and sample dependent, so you may just have to try a lot of different random ones and see which work best.

Next Up Previous Contents

Next: Invariance **Up:** Automatic Speech Recognition **Previous:** The Baum-Welch and Viterbi *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Other HMM applications **Up:** Automatic Speech Recognition **Previous:** Network Topology and Initialisation

Invariance

Although not as up to date as possible, it is as well to remember that VQ and HMM is the staple of single word recognition systems and rather more. It works reasonably well provided that the system is trained on the speakers it is proposed to use it on; this makes it a dead loss for selling to banks to allow people to tell an automatic system their mastercard number. Even a modest amount of speaker independence has to be purchased by training on a whole raft of people and hoping that your speaker is in there somewhere, or at least a clone or close relative of the same sex. The standard *el cheapo* boxes for single word recognition which you buy for your PC or MAC work this way. They often solemnly advertise that the system will only respond to your voice, as though this is a great virtue, but in fact it doesn't even do that particularly well.

I have mentioned one way of improving the VQ side of this approach. Another is to try to explicitly recognise the gaussian mixture model is an estimate for a continuous *pdf* over the states. The only difference in practice is that instead of choosing the closest gausian distribution, we may want to take account of the weights associated with it and choose likelihoods. This yields a semi-continuous Hidden Markov Model. There are some simple ways to improve the HMM side of it too: one can go further and try to model the probability density function for trajectories in the space by directly training a gaussian mixture model on the data consisting of graphs of repetitions of utterances of a single word, and then computing likelihoods for each new trajectory relative to the family of models. After all, it is the set of trajectories with which one is concerned, this is the data. The natural desire to model the *pdf*s for the different words is thwarted in practice by the fact that there are a number of transformations that are done to the data, and this means we never have enough data to get sensible estimates. It is rather as if you wanted to tell a letter A from a letter B when they had been scribbled on a wall, by storing information about which bricks were painted on. You don't get much help from the replication of A's if they occur in different places, when represented in this form, whereas if you had been able to get a description which was shift invariant, it would be possible to pool the data provided by the different A's and B's.

But that requires knowing the transformation we want to factor out, and in general we don't.

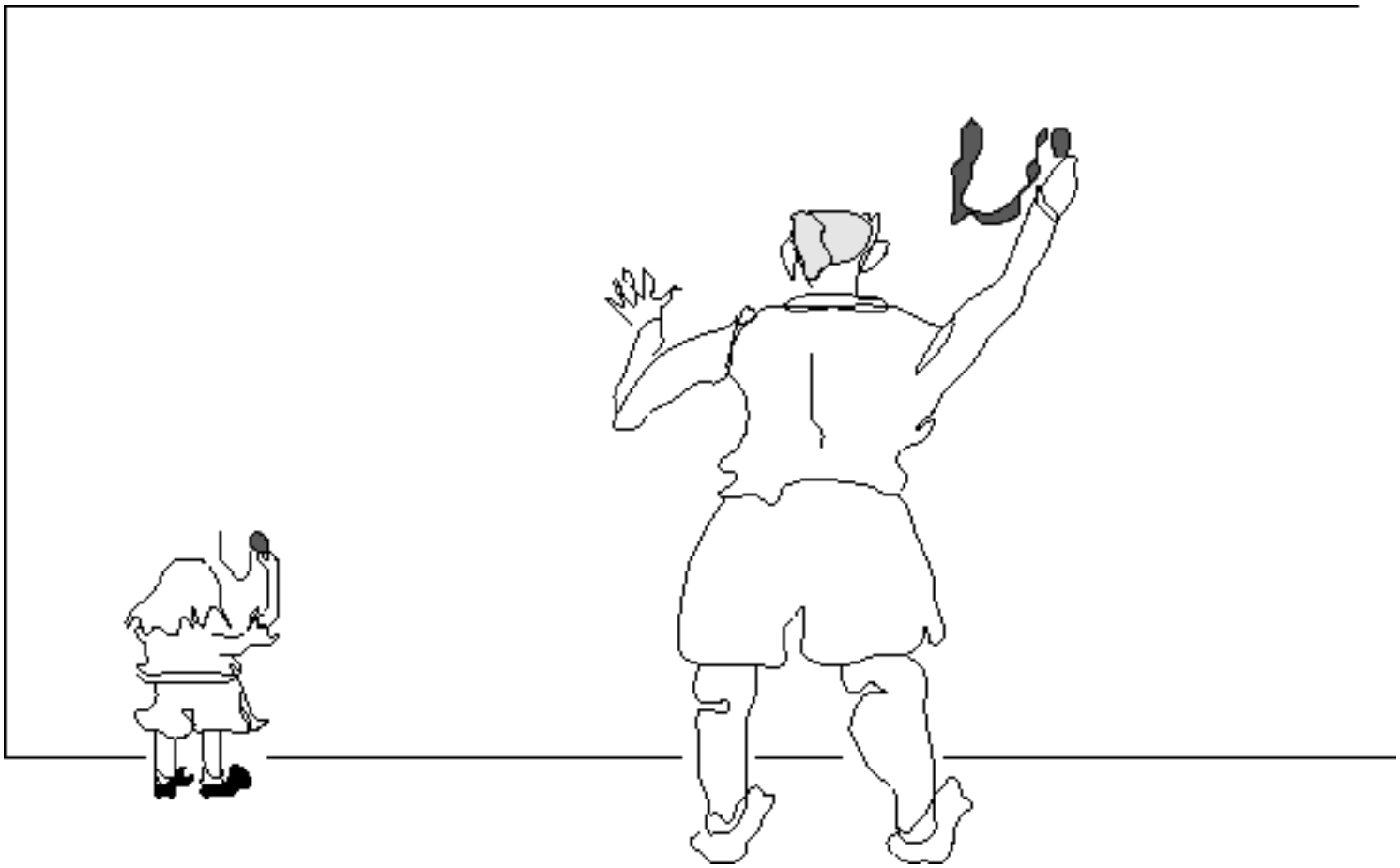
The only thing the prudent human being can do to make life easier for his automatic system is to build in all the invariance he expects to be useful.

One thing, then, that ought to make the ASR practitioners nervous is the big, fat, fundamental assumption that phonemic elements are regions of the space.

To see what other possibilities there are, consider the problem of asking a small girl and a large man to say the words `yes' and `no'. They have different sized and shaped vocal tracts and it is clear as day that their utterances will occupy very different regions of the speech space. Had we given them a piece of chalk apiece and asked them to write the words upon a wall instead of speaking them, we could reasonably have expected that the two dimensional trajectories of the movement of the chalk would also

have occupied different parts of the space. Trying to distinguish the words `yes' and `no', when written down, by looking to see which bricks they are written upon would not, on the face of things, be a good way to go.

Figure 6.2: Translation and scale invariance.



It is clear that the problem arises because the fundamental assumption is clearly wrong in the case of handwritten words, and it might be just as wrong for spoken words too. It isn't the regions of the space you pass through, it's the shape of the trajectory, in some scale invariant way, that determines what a written word is. The HMM if used on the bricks of the wall to determine written words might more or less work for one little girl and one large man, but would flip when confronted with an intermediate size writer.

The problem is that the written word is invariant under both shifts and scaling. And the spoken word is also invariant under some part of a group of transformations: simply making the sound louder throughout will not change the word. Lowering or raising the pitch, within reason, will not change the word. And it is easy to believe that continuous enlargement of the vocal tract won't change the word, either. Adding white noise by breathing heavily all over the microphone, as favoured by certain pop-stars, might also be said to constitute a direction of transformation of a word which leaves the category invariant.

If one excises vowels from the speech of many different speakers, plots the short segments of trajectory

which one then obtains as short sequences in $\mathbb{R}^{16}$, and then projects down onto the screen of a computer, one sees what looks like a good approximation to a seperate gaussian cluster for each vowel. But the cluster has quite a large extension, close vowels sounds appear to overlap, and the distance between centres is not as big as the extension along the major axes. It is tempting to try to split the space into a part which differs in what vowel is being articulated, and an orthogonal part which tells you who is speaking, how loudly, and how hoarse they are, along with much other data which seems to be largely irrelevant to working out what is being said. It would be nice to try to decompose the space in this way, and it is possible to attack this problem, if not cheap.

Consider, as another basis for doubting the fundamental assumption when applied to speech, the various sign languages used by deaf people and others. These consist of stylised gestures of various sorts, performed in succession, at rates comparable with ordinary speech as far as the rate of transfer of ideas is concerned. The question is, are the target gestures actually made in Sign? It can be plausibly argued that they are hardly ever actually `stated' except in teaching the language, and that fluent signers only sketch out enough of a gesture for the signee to be able to decode it, after which they move on rapidly to the next gesture. So signed `words' are idealisations never actually attained. Moreover, the space of gestures references such things as heads and faces, and is thus not readily reducible to trajectories in an absolute space.

Speech has been described as `gestures of the vocal tract', and we have to consider the possibility that it resembles sign languages in its trajectory structure rather more than meets the eye. If so, the crudity of the VQ/HMM model becomes embarrassing. One would have to be desperate to suggest this approach to sign languages. In fact the problem of automatic reading of sign via a camera is an interesting if little explored area for research which might well illuminate other languages. Perhaps the difficulties are simply more apparent for sign languages than for spoken ones.

In the case of speech, it is generally argued that the path is what is wanted, not the trajectory, since going over the same part of the space at a diferent rate will not change what is said, only how fast you say it. This gives another family of transformations of the space of trajectories which would, if this view is correct, leave the word itself invariant.

In general the problem of speaker independent speech recognition is a profoundly difficult one and one which still needs critical thought, as well as more creative approaches.

For confirmation of some of the pessimistic observations of this section, see the program *fview* by Gareth Lee on the ftp site ciips.ee.uwa.edu.au, referenced in the bibliography. Pull it back and play with it: it is the best free gift of the decade.

Next Up Previous Contents

Next: Other HMM applications **Up:** Automatic Speech Recognition **Previous:** Network Topology and Initialisation *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Connected and Continuous Speech **Up:** Automatic Speech Recognition **Previous:** Invariance

Other HMM applications

A Hidden Markov Model is merely a device for compressing strings, otherwise a stochastic grammar. It does not behave well when there is much noise in the strings, symbols which don't occur very often and which have no significance, and it does not do a particularly good job of extracting the brief and conceivably critical transitions which may well characterise, say, stop consonants.

Alternatives, such as simply finding substrings which recur, up to some matching criterion, can be employed and are much faster. Local grammars and variants of them designed to accommodate noise conveniently may be used with only a small amount of ingenuity, and these will be discussed in the next chapter.

HMMs have been used in handwritten character and word recognition; one might be forgiven for suspecting that the choice is dictated by the crudest forms of research practice which requires absolutely no understanding of what you are doing: you copy out an algorithm used for tackling one problem and use it for tackling one which is sufficiently similar to offer hope. There are more intelligent ways which will be discussed later.

Next Up Previous Contents

Next: Connected and Continuous Speech **Up:** Automatic Speech Recognition **Previous:** Invariance *Mike Alder*
9/19/1997

Connected and Continuous Speech

Recognising one spoken word is a start, but it is a long way from being able to dictate to your computer and have it type out what you thought you said, or as any good secretary will, what he or she thinks you think you ought to have said. Systems such as the IBM *Tangora* can recognise a sentence of words from a fifty thousand word vocabulary providing each is spoken with a short space between them. This is because there are a lot of ways of trying to decode continuous speech. The reader should try saying each of the four sentences following out loud a few times:

Ken ewe won ah star net.
Car knew one a stair nit.
Care new oner sten at.
Can you understand it?

The last is the most likely fit to an actual utterance in English, but the earlier versions are likely to be better from an acoustic point of view. In order to have good grounds for preferring the last, one needs to know something of the allowable syntax of the spoken language. The constraints, given by a stochastic grammar of some sort, a kind of statistical model for the strings of the language, are needed in order to control the horrendous number of possibilities. And even working out what the grammar for spoken language *is* can be rather trying. For instance, the most common word in the spoken English language is `uh' or `ah' or `um' or `er'. This is a transactional word which means, roughly, `shut up, I haven't finished talking yet'. It does not occur, for obvious reasons, in the written language at all. So when a system has had the constraints of the written language put in and then somebody says to the system `Tell me about, uh, China', the system may well start telling him about America, since there is no country called Uhchina and the system `knows' it has to be a proper name in its data base beginning and ending with a schwah. There is an apocryphal tale about such a system which asked the user which day he wanted to come back and he sneezed. After thirty seconds of agonising the system replied: `We're closed Saturdays'. So the business of using constraints to determine which word string was used can run into troubles.

The basic problem of sentence recognition is that people don't speak in words. There is no acoustic cue corresponding to the white space in text. This leads the reflective person to wonder if we could cope were all the white spaces and punctuation to be removed from English text, something which invites a little experiment.

In general, the attempt to use the constraints of natural language to improve recognition systems have succeeded only because the systems that didn't use them were unbelievably bad. There are many different kinds of information which human beings are known to use, from syntactic at the phonemic level (knowing what strings of noises can occur) to syntactic at the lexical level (knowing what strings of words can occur) to semantic (knowing what the words and sentences mean), pragmatic (knowing what the speaker is talking about) and prosodic (stress: pitch and loudness variation). Workers are still

exploring the issues involved in using these sources of information intelligently. They have been since the seventies; see the many reports on the HARPY and HEARSAY projects from Carnegie-Mellon University. A discussion on the subject of inferring (stochastic) grammars for a language from a sample will be a feature of the next chapter. You have already seen it done once, rather unconvincingly, using the EM algorithm for Hidden Markov Models.

It should be plain to the thoughtful person that we are a long way from telling how to build the ultimate speech recognition system, and that this discussion merely outlines some of the difficulties. The situation is not actually hopeless, but the problem is difficult, and it would be as well not to underestimate its complexity. The AI fraternity did a magnificent job of underestimating the complexity of just about all the problems they have looked at since the sixties: see the reference to Herbert Simon in the bibliography to chapter one. Engineering builds wonderful devices once the basic science has been done. Without it, engineers are apt to want to plunge swords into the bodies of slaves at the last stage of manufacture so as to improve the quality of the blade: sea water or suet pudding would work at least as well and be much cheaper. The business of discovering this and similar things is called `metallurgy' and was poorly funded at the time in question. Similarly, the basic science of what goes on when we hear some spoken language at the most primitive levels is in its infancy. Well, not even that. It's positively foetal and barely post-conception. It's an exciting area, as are most of the areas involved in serious understanding of human information processing, and progress is being made. But for the man or woman who wants to make a robot behave intelligently, abandon now the thought of having a discussion with it and hope, in the immediate future, for it to be able to handle only single words, or sentences with artificial and restricted syntax.

Next Up Previous Contents

Next: Filters **Up:** Automatic Speech Recognition **Previous:** Other HMM applications *Mike Alder 9/19/1997*

Filters

There is a fairly well developed theory of trajectories in $\mathbb{R}^n$, which encompasses both the case of continuous and discrete time, and which is aimed not at classifying the trajectories but at predicting, interpolating and smoothing them. This entails some kind of modelling process for the production of the function. It is reasonable to care about this sort of thing even if we are interested in classifying trajectories and not predicting, interpolating or smoothing them, for two excellent reasons. The first is that we may reasonably expect to be plagued with noise in whatever dynamic pattern we are faced with, and the theory of filters gives some advice on what to do about this. The second is that any recognition or classification will surely involve us in some kind of modelling of the process which generated the trajectories, and so it is worth looking to see what kinds of models other workers concerned with dynamic patterns have thought worth investigating. We may be able to pinch their ideas.

In the case $n = 1$ we are talking about predicting, interpolating or smoothing the graph of a function of a single real variable, as they used to call it (and still do in some places). The function is usually referred to as a *signal* or a *time series*, depending on whether your source of information is an engineer or a statistician. The business of finding a function which looks to be a good fit to the given graph is called *filtering* if you are an engineer, *regression* if you are a statistician, and *function-fitting* or *function approximation* if you are just one of the boys.

Conceptually, you are confronted with a sequence of dots, or possibly a wiggly line, on a piece of paper which invites you to take a thick, black brush and to draw a smooth curve through the dots or the wiggly line, not necessarily through each dot, maybe passing in between dots or wiggles. Then you can extend your smooth curve with a flourish (extrapolation or prediction), fill in between points (interpolation) or decide you would prefer your curve to the actual data at some particular x value (smoothing). To do this by eye is to invite confrontation with anybody else who did the same thing, since the two smooth curves will surely differ, at least a bit. This leaves us with the problem of automating the process of getting the smooth curve, of getting the *right* smooth curve, or at least one which can be agreed upon as acceptable and also computed automatically. After chapter three, it probably won't come as much of a shock to discover that there is no one `right' solution, and doing it by hand and eye and some dash may be performed with a clear conscience. This method is no less rational and defensible than any other; the rationales for the other methods are more complicated but ultimately hinge on you buying some untestable assumption about the data. The advantage of automating the process is not that you can feel any more confidence in its rightness, but that it can be then done to a lot more data than your arm could cope with.

There are two traditions in the area, one is the statistical tradition (time series) going back to Yule and Kolmogorov which talks of AutoRegressive Moving Average (ARMA) models for time series, and the engineering tradition which talks of Finite Impulse Response and Infinite Impulse Response (FIR and IIR) filters for signals. These are the same thing. I have heard tell of a statistician who asked an engineer

if he wanted ARMA modelling done in his course on stochastic time series for engineers, and was told `no, we don't need that, just see if you can fit in a bit on FIR and IIR filters, if you have time.' Not knowing what these were, the statistician simply shortened the course.

The engineering terminology confuses the objectives with the methods and details of the source of the problem, that is to say it is cluttered with massive irrelevancies and makes it hard to see the wood for the trees. The statistical terminology by contrast has been made so abstract by some (the French school) that you need a strong background in analysis before you can understand what the subject area is, by which time you may have lost interest.

The basic ideas then, have been obscured by the language in that traditional manner so widely believed to be the main contribution of Mathematics to Science and Engineering. It will sadden the reader to learn that a whole machinery of analysis of speech, Linear Predictive Coding, depends upon it. The reader is advised that if she lacks the training in either of the areas mentioned, she will find her patience sorely tried by the literature. I shall try in this section to explicate the main ideas while avoiding details. As you were warned in the introduction to this chapter, the area is a large one, and the most useful contribution I can make in this book is to sketch out the central ideas in my inimitable way, so as to allow you to decide if you want or need to go further, and to suggest where to look if you do.

- Linear Systems
- Moving Average Filters
- Autoregressive Time Series
- Linear Predictive Coding or ARMA modelling
- Into $\mathbb{R}^n$
- States
- Wiener Filters
- Adaptive Filters, Kalman Filters

Next Up Previous Contents

Next: Linear Systems **Up:** Continuous Dynamic Patterns **Previous:** Connected and Continuous Speech

Mike Alder
9/19/1997

Linear Systems

Definition 13973

A *discrete real time series* is a map from $\mathbb{Z}^+$ or $\mathbb{Z}$ into $\mathbb{R}$; we think of the integers as consecutive ticks of a clock, and the real values u_n or $u(n)$ as being the value of some physical variable at the n^{th} clock tick.

For example, the clock ticks might be once a day and the values might be the Dow-Jones averages on those days, or the clock ticks might be once every twentieth of a millisecond, and the values the voltages output by a microphone as you talk into it. Some notations have $u(n)$ for the value and hence u as the time series, which is evidently a function; others write the sequence $\{u_1, u_2, \cdots, u_n, \cdots\}$ which is clumsier but more familiar to the older generation. I shall use the function language because it is neater and cuts down on the number of different kinds of things you have to think about.

Definition

A *continuous real time series* is a continuous map from $\mathbb{R}$ or $\mathbb{R}^+$ into $\mathbb{R}$. We think of the input as being continuous time and the output as being the value of some changing measurement at that time.

Since we are going to be dealing with real life, where it is impractical to take a measurment at every conceivable instant of time, we have no immediate interest in such things except that we may like to believe for philosophical reasons that a discrete time series came from sampling a continuous one. Of course, you have to be pretty committed to metaphysics to believe that using $\mathbb{R}$ for time or space is anything more than a linguistic convenience. Maybe space-time comes in very, very small lumps.

It is common to look at stock exchange figures and to feel that there is altogether too much wild variation in consecutive values. A standard way of *smoothing* the stock exchange time series is to replace the central value on a day by the average of the three days consisting of the original day, the preceding day, and the succeeding day. This cannot be done in real time, you get to be a day late with your figures, but predictions on smoothed data might be more useful, so it could be worth it. This is an example of a *moving average filter*; it takes in one time series and outputs another. In fact this is the definition of a filter.

Definition

A *filter* is a map from a space of time series to a space of time series.

The usual application is to filter out random noise, as in the moving average example. In the old days, when life was hard and *real* engineers had to do three impossible things before breakfast, the time series was likely to be continuous, was almost invariably a voltage, and a filter was made out of capacitors,

inductances, resistors and similar knobbly things you could pick up and lose. The signal went in, and emerged purified, like water having the dirt filtered out by pouring it through well, through filters. The mental image of a dirty, insanitary time series coming in, and a somewhat improved one leaving, still pervades the language. In these latter days, the time series is more likely to be a sequence of numbers input to a computer by some data acquisition system or output from it to some mechanism, while the filter is likely to be a software construct. In this case they become known as *digital filters*. I shall consider only digital filters here, with a brief aside now and again to indicate the directions for the continuous case.

Next Up Previous Contents

Next: Moving Average Filters **Up:** Filters **Previous:** Filters *Mike Alder*
9/19/1997

Moving Average Filters

The moving average filter as described can be modified in a number of ways: we can take more than three numbers, we can insist that the filter be *causal* which means that the output at time *k* cannot require knowledge of the input at a later time, and the averages can be weighted. If I give you the numbers one quarter, one half, one quarter, with the instruction to add one quarter of yesterdays Dow Jones to half of today's to one quarter of tomorrows, we output (a day late) a *weighted moving average*. The histogram of three numbers, one quarter, half, one quarter, is a sort of blurring mask which we run over the time series to modify it. we can regard it as a function defined from $\mathbb{Z}$ to $\mathbb{R}$ which is zero except at -1,0 and 1, where it takes the given values. You may be reminded of the blurring done by morphology techniques.

Definition 13992

If f and g are functions from $\mathbb{Z}$ to $\mathbb{R}$, the *convolution* of f with g, written $f * g$, is a new function from $\mathbb{Z}$ to $\mathbb{R}$ defined by

$$f * g(n) = \sum_{i=-\infty}^{\infty} f(n-i)g(i)$$

It is clear that I smooth f by performing a convolution with a suitable g. What I do is to compute the new, smoothed function $f * g$ at a point *n* by centring *g* on *n* (that is move the 0 value of *g* over *n*) and then multiply the corresponding values of *f* and *g* and sum them. You might think I have written *g* backwards, but in this form it works if both functions are defined for the non-negative numbers only. A variant with an integral sign for the case of continuous functions defined on $\mathbb{R}$ or $\mathbb{R}^+$ is left as an exercise for the student.

If f is not defined for all values of $\mathbb{Z}$, then I need to do some obvious fiddling, like making *f*(*n*) zero when *n* is negative or changing the limits on my summation. Since filters, to be implementable, must have only a finite number of terms, in practice one is rather restricted to smoothing envelopes *g* which are non-zero for only a finite (and usually pretty small) number of values.

If the values of *g* are allowed to be negative some rather surprising things can happen, and it is useful to experiment by obtaining some time series, writing down a Moving Average filter *g*, and performing the convolution in a program to see what comes out. In particular, it is possible to input white noise and get out a time series which manages to look deeply significant. White noise is, for those who don't know, noise which has equal spectral representation at all frequencies, and is hence entirely mythical. Approximations to it over the audio spectrum sound like surf on a beach and are said to be very soothing

to listen to. Well, more soothing than most things one hears on the radio these days, at least. One can produce quite reasonable simulations of gaussian white noise by choosing, for each clock tick, quite independently, a random number according to a gaussian or normal distribution with mean zero and variance one. The resulting graph or signal looks a mess. If you haven't got a gaussian die to throw, fiddling your random number generator to produce an approximation to one is an exercise which is almost as soothing as listening to the result played back through a speaker, and gives a somewhat greater sense of accomplishment.

If we take it that *g*, the function with which we propose to accomplish a smoothing, consists essentially of a small number of non-zero numbers, we can refer to them as the *coefficients* in the Moving Average filter, and then a filter of order *k* has just that number of coefficients. More accurately, we can take the set of *k* numbers

$$b_0, b_1, \cdots, b_{k-1}$$

and filter u to v by

$$\forall n \in \mathbb{Z}, v(n) = b_0 u_n + b_1 u_{n-1} + \cdots + b_{k-1} u_{n-k+1}$$

so as to get a causal MA filter determined by the filter-vector of coefficients. Moreover, we could extend the idea by having a Moving Average filter which changed as we move along the time axis. It had better not change too fast, or the process is unlikely to be intelligible in terms of changing filters, but if we take a moving average family of filter vectors which change in time, we can use this to model speech. Instead of using white noise as input, we can use a pulse at the frequency of the pitch of a human voice, and argue that what our filter family is doing is modelling the way the vocal tract acts on the vocal chords to produce a sound. This is conceptually fairly straightforward, and gives an idea of a class of models, even if it lacks a certain amount of necessary detail.

Next Up Previous Contents

Next: Autoregressive Time Series **Up:** Filters **Previous:** Linear Systems *Mike Alder*
9/19/1997

Autoregressive Time Series

A time series *f* might have the property that the value of *f* at time *n* is related to the value at an earlier time. In particular if *f* were periodic with period *T* then we would have *f*(*n*) = *f*(*n*-*T*) or perhaps *f*(*n*) = -*f*(*n*-*T*/2). If *f* were built up as a finite sum of periodic functions with a variety of periods and phases, we could write

$$f(n+1) = a_0 f(n) + a_1 f(n-1) + \cdots a_{r-1} f(n-r+1)$$

for some suitable collection of *r* coefficients.

Definition 14028

A time series $f : \mathbb{Z} \longrightarrow \mathbb{R}$ is said to be *autoregressive of order r* iff

$$\exists a_0, a_1, \cdots a_{r-1} \in \mathbb{R} : \forall n \in \mathbb{Z}, f(n+1) = \sum_{i=0}^{r-1} a_i f(n-i)$$

As I have defined it, some series are autoregressive, while most are not. You choose some starting sequence of *r* values for *f*, and then operate upon them to produce the next value of *f*. Now you slide up one value and do the same again. A little thought shows that if the coefficients were of the wrong sort, the series could easily blow up; for example if *r*=1 and we have *f*(*n*+1) = 2 *f*(*n*), we get a geometric progression going bigger, while *f*(*n*+1) = 1/2 *f*(*n*) has it sinking to zero fast. The situation with more coefficients is more complicated and is considered by engineers under the title of *stability of the filter*.

To the engineer, a set of coefficients like the b_i above is just asking to be made into a filter. Is such a thing a filter? If I give you a starting segment you can go ahead and generate a time series, but what do I do if I have a whole time series as input? Well, nothing subtle. I can MA filter it and add it in, but that doesn't make for a separate kind of filter called Autoregressive (AR) filtering, although you might naively think it did.

What I can do is to consider a class of filters, called IIR filters , where if you input a time series, say *u*, you get output time series *v* and where

The IIR is short for *Infinite Impulse Response* and comes from the observation that if you choose *r* to be 1, input a time series *u* which is zero everywhere except at one time where it takes the value 1, make b_0 any non-zero value you wish, then unless the a_i are rather unlikely, the output time series will never become permanently zero (Although it can get pretty close to it). If you put such an input into a MA filter, then of course it rapidly goes to the average of the zero inputs, so is zero except for some finite period of time. Hence the term *FIR filter*, F for Finite, for what I have called a Moving Average filter, because my name tells you a lot more about what it is.

Much of the history of Filtering theory dominates the older treatments in the Engineering texts.

Given a sequence of real or complex numbers, some engineers feel an irresistible urge to make them the coefficients of a complex power series or Laurent series, and use the resulting complex function instead of the series. It is possible to satisfy oneself that there is a one to one correspondence between convergent infinite sequences and complex analytic functions. This leads to turning problems about time series into problems about properties of complex analytic or more generally meromorphic functions. (Meromorphic functions may go off to infinity at some discrete set of points while analytic ones don't.) Having suffered the agonies of mastering complex function theory, they don't see why you shouldn't suffer too; they may firmly believe that such suffering enobles and strengthens the character, although others may doubt this. I should undoubtedly be drummed out of the Mathematician's Union were I to cast doubts on this view. Besides I like Complex Function theory. So I won't.

Next Up Previous Contents

Next: Linear Predictive Coding or **Up:** Filters **Previous:** Moving Average Filters *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Into **Up:** Filters **Previous:** Autoregressive Time Series

Linear Predictive Coding or ARMA modelling

For reasons which are not altogether convincing, it is not uncommon to model speech as if it were produced by IIR filtering a glottal pulse input (for voiced sounds) or white noise (for unvoiced sounds). Then if we know what is supposed to have gone in, we know what came out, we can calculate the coefficients which give the best fit to the output over some period of time. As the vocal tract changes, these coefficients are also supposed to change in time, but relatively slowly. So we change a fast varying quasi-periodic time series into a vector valued time series, or a bit of one, which I have called the trajectory of an utterance. The argument for Autoregressive modelling suggested above hints at relationship with the Fourier Transform, which emerges with more clarity after some algebra.

This approach is called *Linear Predictive Coding* in the Speech Recognition literature.

ARMA modelling with its assumptions, implausible or not, is done extensively. A variant is to take a time series and `difference' it by deriving the time series of consecutive differences, $v(n) = u(n) - u(n-1)$. This may be repeated several times. Having modeled the differenced time series, one can get back a model for the original time series, given some data on initial conditions. This is known as ARIMA modeling, with the I short for Integration.

The modelling of a stationary time series by supposing it arrives by filtering a white noise input is a staple of filtering theory. The method is perhaps surprising to the innocent, who are inclined to want to know why this rather unlikely class of models is taken seriously. Would you expect, say, the stock exchange price of pork belly futures to be a linear combination of its past values added to some white noise which has been autocorrelated? The model proposes that there is a random driving process which has short term autocorrelations of a linear sort and arises from the driving process by more autocorrelations, that is dependencies on its own past. Would you believe it for pork bellies? For pork belly futures? Electroencepghalograms? As a model of what is happening to determine prices or anything much else, it seems to fall short of Newtonian dynamics, but do you have a better idea? Much modelling of a statistical sort is done the way it is simply because nobody has a better idea. This approach, because it entails linear combinations of things, can be written out concisely in matrix formulation, and matrix operations can be computed and understood, more or less, by engineers. So something can be done, if not always the right thing. Which beats scratching your head until you get splinters.

Once the reader understands that this is desperation city, and that things are done this way because they can be rather than because there is a solid rationale, he or she may feel much more cheerful about things. For speech, there is a theory which regards the vocal tract as a sequence of resonators made up out of something deformable, and which can, in consequence, present some sort of justification for Linear Predictive Coding. In general, the innocent beginner finds an extraordinary emphasis on linear models throughout physics, engineering and statistics, and may innocently believe that this is because life is generally linear. It is actually because we know how to do the sums in these cases. Sometimes, it more or less works.

Next Up Previous Contents

Next: Into **Up:** Filters **Previous:** Autoregressive Time Series *Mike Alder*
9/19/1997

Into $\mathbb{R}^n$

Given a time series $u : \mathbb{Z} \longrightarrow \mathbb{R}$, it is sometimes convenient and feasible to turn the time series into a map from $\mathbb{R}^n$ to $\mathbb{R}^n$ which is iterated, with some noise added. All we do is to take a sequence of n consecutive time values of the series as input, and the shift to one time unit later as output. That is, we look at the map

$$\begin{pmatrix} u(t) \\ u(t-1) \\ \vdots \\ u(t-n+1) \end{pmatrix} \longmapsto \begin{pmatrix} u(t+1) \\ u(t) \\ \vdots \\ u(t-n+2) \end{pmatrix}$$

This I shall call the *delay map*.

Similarly if we have a MA filter of order k from a time series u to a time series v we can write this as a linear map from $\mathbb{R}^{k+1}$ to $\mathbb{R}^k$. For example, if we have that

$$v(n) = b_0\, u(n) + b_1\, u(n\text{-}1)$$

we can write

$$\begin{pmatrix} v(n) \\ v(n-1) \end{pmatrix} = \begin{pmatrix} b_0 & b_1 & 0 \\ 0 & b_0 & b_1 \end{pmatrix} \begin{pmatrix} u(n) \\ u(n-1) \\ u(n-2) \end{pmatrix}$$

Writing such a map as f, it is not hard to see that ARMA modeling of v becomes a modeling of f by matrix operations. This approach has the additional merit that if we have a time series of vectors, we can simply amalgamate the vectors in the same way, and the general theory doesn't trouble to distinguish between the cases where a time series is of vectors having dimension one and where it has any other dimension.

It is not particularly reasonable to expect the delay map to be linear or affine, with or without noise, but we can keep our fingers crossed and hope that it is. Chaotic systems arise rather easily when it isn't. Our position is not so much that linear and affine systems are the ones which arise in practice a great deal, it's that those are the ones we can figure out what to do something about.

If we take a point in the domain of the delay map and apply the map to this point, and then to the result

of applying the map, and thus iterate the map on this point, we get a set of points in $\mathbb{R}^n$. We may compute the mean and covariance matrix for this set in order to get a second order description of it, just as for a set of pixels in $\mathbb{R}^2$. It is called the *autocovariance matrix*. It may turn out that the set of points lies on a subspace of dimension significantly less than *n*. Anything that can be said about the set of points obtained by parcelling up the time series in this way is a statement about the process which generated the time series, so we have a distinct interest in describing point sets in $\mathbb{R}^n$.The mean and covariance matrix are rather inadequate descriptors for complicated shapes, and so it is possible to go to higher order. It is also possible to go to local descriptions which are locally low order and to piece them together. We shall have more to say about this later.

The simple minded trick of taking a time sequence of numbers and turning it into a time series of vectors by putting a moving window over the time series in the hope that scrutiny of the resulting point sets will be informative is also done in the old piecewise affine neural nets, where a simple perceptron then becomes a MA filter with a non-linear threshold stuck on the end. These are called *time-delay* neural nets. We can also feed the output back into the input through a subnet which procedure yields a *recurrent neural net*. The range of basic ideas here is really fairly limited.

Next Up Previous Contents

Next: States **Up:** Filters **Previous:** Linear Predictive Coding or *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Wiener Filters **Up:** Filters **Previous:** Into

States

The assumption that we are using at present is that it suffices to study time series with linear dependencies so that if we take vector values or time blocks or both, we have an output vector valued time series *v* produced by an input vector valued times series *u* (which might be uncorrelated gaussian white noise or might not) and with an autoregressive component. Formally,

$$v(n+1) = A\, v(n) + B\, u(n)$$

for vectors $u(n), v(n), n \in \mathbb{Z}$, and matrices *A*,*B*. It is usual to introduce another complication by

supposing that there is an internal state which is produced by such a process and that this is observed by some means which may introduce more noise. Thus we may have two equations:

Here, *z* is some internal state which is not observed, it is `hidden', and which gives rise to *v* which is observed. The output vectors lie in some affine subspace of the output space in general, and finding it and its dimension is a part of the problem of filtering the process given by the above equations.

In even more generality, we may have the case where the matrices A,B,C,D are allowed to change in time, although not, one hopes, too quickly.

Next Up Previous Contents

Next: Wiener Filters **Up:** Filters **Previous:** Into *Mike Alder*
9/19/1997

Wiener Filters

To take the simplest example of filtering, suppose $\{u = u(n) : n \in \mathbb{Z}\}$ is an input series and $\{v = v(n) : n \in \mathbb{Z}\}$ is an output series. I want to have a convolution of *u* with some finite filter so that *u* looks like *v*. Let us suppose that I have some finite amount of history of the past of *u*, say for $j \in [1 \ldots n]$. Then I want to choose some coefficients $b_0, b_1, \cdots, b_k$ so that

$$\sum_{j \in [1 \ldots n]} (v_j - b_0 u_j - b_1 u_{j-1} - \cdots b_k u_{j-k})^2$$

is a minimum.

If I take *k* to be 1, for purposes of exposition, then I am trying to minimise

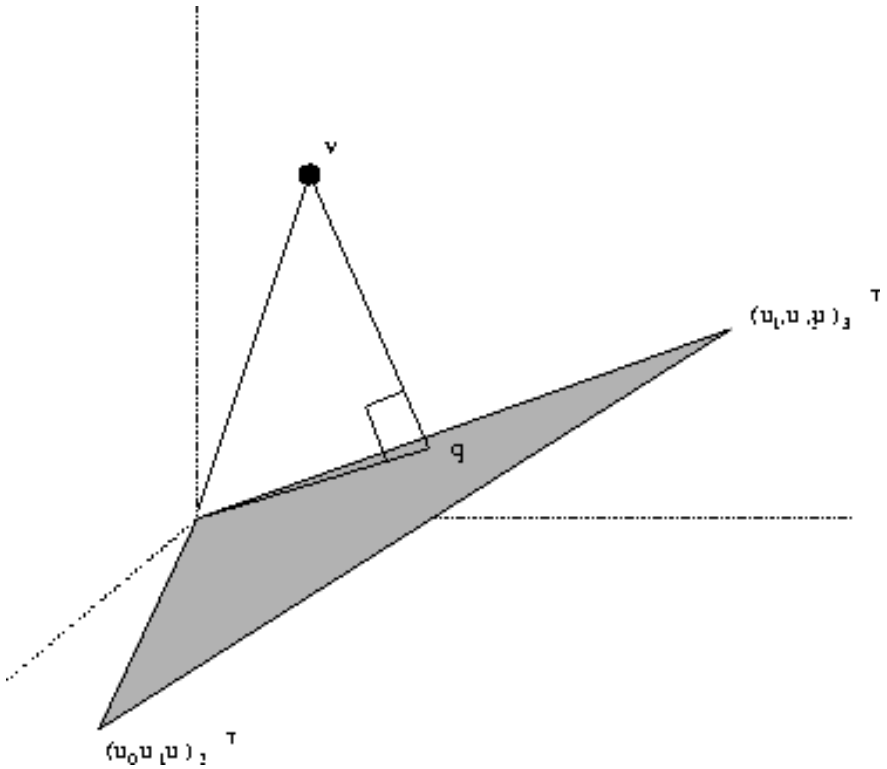
$$\sum_{j \in [1 \ldots n]} (v_j - b_0 u_j - b_1 u_{j-1})^2$$

The expression is best seen as a distance in $\mathbb{R}^n$ from the point $\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ to the plane

$b_0 \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} + b_1 \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{pmatrix}$ determined by all possible choices of b_0 and b_1. In general, it will be a *k*-dimensional subspace determined by all possible choices of the b_i.

The problem of finding the closest point on a plane, *P* to a given point *v* in $\mathbb{R}^n$ but not on the plane, is a simple piece of linear algebra. We observe that if the closest point to *v* on *P* is called *q*, then the line joining the point *v* to the point *q* is orthogonal to the plane. Hence it is orthogonal to the vectors which span the plane. This follows immediately from Pythagoras' theorem, although it is called rather pompously the `orthogonality principle' in many text books. If you draw a picture for *n*=3 you should have no particular difficulty with this: see *Fig.* 6.3.

Figure 6.3: Closest point to a plane through the origin: line **v q** orthogonal to plane.



For orthogonal vectors the dot product is zero, so we obtain the two equations

$$\left(b_0 \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} + b_1 \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} - \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \right) \bullet \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = 0$$

and

$$\left(b_0 \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} + b_1 \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} - \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \right) \bullet \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} = 0$$

If instead of having a filter of order 2 we had used k coefficients, we should have had the same orthogonality condition (still by Pythagoras' Theorem!) but this time there would have been k of them. $\bullet$ denotes the dot product in $\mathbb{R}^n$.

The k equations are called the *normal equations* and solving them gives the point q. They are simply k linear equations in k unknowns. We leave it to the reader to decide when there is a unique solution.

In the filtering case, the equations are also known as the *Yule-Walker* equations, and the corresponding equations in the continuous case are called the *Wiener-Hopf* equations. The solution gives a so-called *optimal filter* but it should be noted that we are simply trying to get as close as we can to a known point in $\mathbb{R}^n$ while staying in a linear subspace.

An obvious practical objection to the process as described above, is that the filter pays as much attention to the current values as it does to the remote past ones. In other words we have chosen a particular (Euclidean) metric on the space $\mathbb{R}^n$ given by

$$\left(d\left(\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \right) \right)^2 = (x_n - y_n)^2 + (x_{n-1} - y_{n-1})^2 + \cdots + (x_1 - y_1)^2$$

where it might be more realistic to discount the past by weighting it less. This can be achieved by changing the metric to:

$$\left(d\left(\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \right) \right)^2 = (x_n - y_n)^2 + \frac{1}{2}(x_{n-1} - y_{n-1})^2 + \cdots + \frac{1}{2^{n-1}}(x_1 - y_1)^2$$

This can be done by defining a new inner product. The choice of $\frac{1}{2}$ raised to increasing powers can be varied to taste. A new inner product can be conveniently represented by a symmetric, positive definite matrix. If $\mathbf{x}$ and $\mathbf{y}$ are vectors and *A* is such a matrix, the new inner product will be $\mathbf{x}^T A \mathbf{y}$ and the good old `dot' product is simply the special case where *A* is the identity matrix. The above equations may look slightly different in this form.

It is common to rewrite the (*k*+1) normal equations as follows:

$$\forall j \in [0 \cdots k], (b_0 \mathbf{u}_n + b_1 \mathbf{u}_{n-1} + \cdots b_k \mathbf{u}_{n-k}) \bullet \mathbf{u}_{n-j} = \mathbf{v} \bullet \mathbf{u}_{n-j}$$

or alternatively as:

$$\forall j \in [0 \cdots k], \sum_{i=0}^{k} b_i (\mathbf{u}_{n-i} \bullet \mathbf{u}_{n-j}) = \mathbf{v} \bullet \mathbf{u}_{n-j}$$

If you reflect on what *n* ought to be in practice, you can see that it should be as long a sample as you have from starting time to finishing time, except for the small problem of shifting outside the region for which we have data. A certain amount of wastage will occur at the ends of a finite sample. Note that we do not in fact need the time series $\mathbf{v}$ we need only its dot product with different shifts of the time series $\mathbf{u}$.

A slightly different formulation can be obtained by dividing the above equations on both sides by *n*; in this case we can

define the *Auto-Correlation Function (ACF)* for a time series *u* as the function which assigns to an integer *j* the average or expected value of *u*(*n*)*u*(*n-j*).

We can define the *Cross-Correlation Function (CCF)* for *u* and *v* to be the average or expected value of *v*(*n*) *u*(*n-j*). If the time series *u* is *stationary* then

$$\frac{1}{n}(\mathbf{u}_{n-i} \bullet \mathbf{u}_{n-j}) = r_{11}(i-j)$$

where $r_{11}(t)$ is the ACF of *u* at *t*. Similarly

$$\frac{1}{n}\mathbf{v} \bullet \mathbf{u}_{n-j} = r_{12}(j)$$

where $r_{12}(t)$ is the CCF of *v* and *u*. Whereupon the Normal-Yule-Walker-Wiener-Hopf equations become:

$$\forall j \in [0 \cdots k], \ \sum_{i=0}^{k} b_i r_{11}(i,j) = r_{12}(j)$$

with the obvious changes made for the case when everything is continuous.

The expected value of the CCF of two functions may be zero; if *v* is a signal and *u* is random noise, then this should be the case, and if *u* is uncorrelated white gaussian noise with zero mean, then the ACF of *u* (the CCF of *u* with itself) ought to be zero except at 0 where it should be the variance. This is the sort of thing which is dead obvious, but when said in algebra can leave you wondering whether it is profound or not.

Not.

If we write the time series as a time series of vectors obtained by storing consecutive windows of *n* of them and moving the windows along to turn the time series of real numbers into a time series of vectors in $\mathbb{R}^n$, as described earlier, then we can compute the autocorrelation matrix of a piece of the time series regarded as a set of vectors. The autocorrelation matrix is the covariance matrix with a shift, because the mean is not subtracted from the values when you compute the autocorrelation matrix. It is easy to verify that the matrix equations relating the covariance matrix with the autocorrelation matrix are

$$Cov(Data) = AutCor(Data) - \frac{N}{N-1} M M^T$$

where *Data* is the matrix formed by listing N vectors in $\mathbb{R}^n$ to give an $N \times n$ matrix, *M* is the $n \times 1$ matrix of the mean of the data vectors, M^T is its transpose, and *Cov*(*Data*) and *AutCor*(*Data*) are self explanatory.

Next Up Previous Contents

Next: Adaptive Filters, Kalman Filters **Up:** Filters **Previous:** States *Mike Alder*
9/19/1997

Adaptive Filters, Kalman Filters

This brief discussion of adaptive filtering is intended to place the general ideas of what they do and how they do it in some sort of context relevant to pattern recognition. The details of how to do the sums are easily obtained from the sources given in the bibliography, should that look to be a good idea.

The last subsection should have made it clear that computing the values of an optimal MA filter was essentially the same idea as fitting a straight line through a set of points. After all, if I give you data consisting of $\{(x_1,y_1),(x_2,y_2),(x_3,y_3)\}$ and ask for the line $y = mx+c$ that fits the data best,

then you follow Gauss (by about two hundred years; my how time flies when you're having fun) and simply take the sum of squares:

$$(y_1 - (mx_1 + c))^2 + y_2 - (mx_2 + c))^2 + (y_3 - (mx_3 + c))^2$$

and choose m and c so as to make this a minimum. This sum of squares we regard as the distance from the point $\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$ to the plane $m \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + c \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. This is a minimum at a point $\mathbf{u}$ on the plane

when the line joining $\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$ to the point $\mathbf{u}$ is orthogonal to the plane and hence to its basis

vectors. This is precisely what we did with a filter having two coefficients. Gauss was confronted with the problem of continuously updating his estimate; he had the batch mode least squares worked out by 1795, and devised an online least squares system quarter of a century later. He also did rather a lot of other things in the same period.

If I give you a stream of (x_i,y_i) and leave you to find the best fit straight line to the data, you can wait until the stream stops and then start working or you can take a block of some number, do it for the block, and then work out how to pool your results with those for the next block of data. A block of data can of course consist of just one datum. If you have reason to suspect the best fit line may be changing in time, you may want to discount old data somewhat. When the model does not change its parameters in time we say it is *stationary*, a definition which raises as many questions as it answers, and a recursive least squares best fit algorithm which can discount the remote past can track a non-stationary process.

The various methods of doing adaptive filtering, are, in effect doing just this. The details can be found in some of the books given in the references.

In Kalman filtering the process is taken to a high level. In the simplest case, the equation $y= mx+c$ is generalised so that y,x and c are vectors and m is a matrix, and c is often taken to be gaussian white noise that has been filtered by a MA filter to autocorrelate the noise. It remains true however that except for the complication of an inner product other than the old dot product to describe the varying degree of credibility of past data and the autocorrelations of the noise with time, what we have is basically a least squares best fit problem done sequentially rather than in batch mode. The choice of a modified least squares can either be justified by placing restrictions on the model class (which are never in practice confirmed to be correct), or not justified at all and regarded as a simple and workable heuristic (as Gauss did) for getting an answer of some sort which is justified *post hoc* and traded in for a better method when one turns up. Statisticians seem to be tempted by the first approach; they feel the comforts of philosophy keenly. Engineers, as is well known, incline to be brutal, insensitive oafs who can't distinguish clearly between rationales and rationalisations and don't trust either.

Kalman filtering is of considerable interest partly because instead of the vector *c*, above, being a noise vector to be got rid of, it may also be interpreted as a control function to be used to force the system from one state to another, so there is a theory of control which is dual to the theory of filters. This halves the amount of work you have to put in to understand a given amount of material, which is a *good thing*.

Next Up Previous Contents

Next: Fundamentals of dynamic patterns **Up:** Filters **Previous:** Wiener Filters *Mike Alder 9/19/1997*

Fundamentals of dynamic patterns

Let us suppose that, as with Speech, we have decided what *n* things to measure, and have made a sequence of measurements for each of a number of events, such as will certainly occur if we are attempting to recognise words. We may suppose that these have been labelled appropriately, and next wish to take a new, unlabelled such trajectory in $\mathbb{R}^n$ and classify it. What basic issues occur? What considerations need to be investigated, whether the problem is in speech or in any other form of dynamic pattern recognition?

The first consideration is that of invariance: if there are known to be groups or parts of groups of transformations under which the trajectory classification is invariant, this information needs to be taken into account. If, for example, the trajectory classification is invariant under a monotone reparametrisation of the path, then we can reduce each trajectory to a canonical time and rate by reparametrising by path length. If there are operations on the space $\mathbb{R}^n$ under which the classification remains invariant, then the transformations should be factored out. If, for example, adding any value to one co-ordinate does not change the classification, then simply project down onto the other co-ordinates and ignore the one that doesn't make a difference. It can only complicate the recognition. Of course, there ought to be some good grounds for believing that the invariance is as claimed, if it is not, you could be making big trouble for yourself.

The main reason for wanting to factor out invariance of as many sorts as possible is, as has been mentioned above, to get more data into the recognition system. To make this clear, if you have a hundred printed digits in each category from 0 to 9 *but they can be anywhere on the page* and if you didn't register them in some way, you would be obliged to learn the (shift) invariance, and this would be wasting data. Similarly, if you had them in different sizes but otherwise identical, it would be sensible to normalise them. This way, you increase the effective size of your training set by reducing the size of your parameter space.

In practice we may be obliged to learn the invariance because we don't know what it is in advance.

This may be done more or less efficiently. Starting out with the intention of finding out what the invariants are as a conscious aim is hard to beat as a prelude to an efficient way of finding them; relying on chance or the kindness of the gods is not recommended.

To focus ideas, suppose we have measured some properties of an aeroplane silhouette which describe the shape in some way. For the present we could suppose that it was done by computing moments of the set of pixels shown in *Fig.* 6.4

Figure 6.4: An aeroplane silhouette.

Let us suppose we have the plane coded as a point in $\mathbb{R}^n$ for some reasonably large $n \gtrsim 20$. I shall assume that no invariance has been built into the measurement process. Now suppose we rotate the figure about its centroid by some small angle and measure the resulting object as a new point in $\mathbb{R}^n$. If the change is sufficiently small then we may expect that the resulting point will be close to the original point. Continue in this way with a sequence of very small rotations, and the measurement process will embed the Lie group of rotations, $SO(2,\mathbb{R}) \cong S^1$ in $\mathbb{R}^n$. If the aeroplane had some symmetry under rotation, for example if it were a square cross, then the map of the circle group S^1 into $\mathbb{R}^n$ would not be an embedding, the result of rotating by $\frac{\pi}{2}$ and integer multiples thereof would take the object to the same point in $\mathbb{R}^n$. For objects not having rotational symmetry, the map embeds the one dimensional manifold of rotations, otherwise, as has been said already, known as the Lie group SO(2) and looking uncommonly like the circle $S^1 = \{z \in \mathbb{C} : ||z|| = 1\}$, into $\mathbb{R}^n$. Similarly, if we scaled the object by different amounts between, say, 60% and 140%, we would be embedding a copy of the interval [0.6, 1.4] in $\mathbb{R}^n$.

And if we generated some shifts it would embed a neighbourhood of the origin in $\mathbb{R}^2$ in $\mathbb{R}^n$, and if we did the lot we should have a four dimensional manifold embedded in $\mathbb{R}^n$ by the measurement process.

The embedding would not generally be flat, so the dimension of the embedded set might be four, but it could normally be expected to go outside a four dimensional affine subspace; after all, the dimension of the circle is one, but it would be quite normal to find an elastic band taking up a two dimensional piece of space, usually a bit of the carpet.

Now if we have a lot of different aeroplanes rotated, shifted and scaled, we might want to work out which aeroplane we were looking at, or we might want to say something about the shape of the manifold of transformations. How could we detect the circle embedded in the enclosing space by the rotations? Remember, all we have is a finite sample of points. Well, the dimension is one, which means that locally we would find that the set of points would lie along a line. If we were to take the set of points obtained by rotating the aeroplane, choose one of them, take a small ball in the space and compute the covariance matrix for the set of points in the ball, then provided there are enough points in the ball and the embedding is reasonably smooth, we would find that all the eigenvalues except for the largest were very close to zero. *And this would be true for any initial such starting point.* The number of non-negligible eigenvalues would tell us something about the dimension of the transformations in general.

This procedure is generally used for dimension reduction, not usually locally, and is known, *inter alia*, as *Principal Component analysis.* It is essential to do it locally if we want an honest estimate of the dimension of a non-flatly embedded manifold. If the dimension is not the same in different parts, it isn't a manifold. Many transformations may be expected to arise from manifolds, (Lie groups, or neighbourhoods thereof) although not all.

If vowel sounds are excised and embedded in a space of simulated filterbank coefficients, it is found that the set of trajectory fragments corresponding to a single vowel is pretty much a gaussian cluster in the space, the dimension of which is less than the enclosing space. The centres of the vowels lie in a lower dimensional region- almost a plane in fact, and the locations of the centres projected onto this plane look just like the diagrams of the vowel space which phoneticians draw. This suggests that the space of vowels is two dimensional, that it is nearly but not quite embedded flatly in the filterbank space, and that the variation between vocal tracts and patterns of enunciation occurs in a low dimensional complement of this space. Confirming this would be of some potential value in Speech Recognition, because one could normalise trajectories by shifting them and projecting them onto the appropriate manifold; the projection would not in general be a simple matter of finding the closest distance however, which is all that has been tried so far.

This method allows us to estimate the dimension of the transformations under which the classification is invariant, provided we have enough data. Indeed, if a new vocal tract comes along which is a transformed version of one we have heard before, and if we hear enough identifiable phonemic elements, we might learn the transformation and then apply a learned transformation. For example, by learning part of a scaling transformation on aeroplane images, it has been found possible to extrapolate from a learnt range of 60% to 140% of the size of an image, down to about 20%, at which level resolution becomes an issue. The sudden transition from aeroplane images with different amounts of scaling to squeaky voices with different amounts of squeak will not alarm the reflective reader. We are talking about the same thing abstractly. I shall discuss this at greater length when we come to syntactic pattern recognition of which dynamic pattern recognition is a special case. In the case where the embedding of the transformation group is flat, we may not only compute its dimension, we can factor it out of the

measurement space altogether by taking a linear transformation of the variables we use to measure the state of the system, and then neglect the last *r*, where *r* is the dimension of the transformation group. If the transformation group is embedded in a non-flat way, which is regrettably the normal situation, we may seek for a non-linear transformation of the space which will accomplish the same thing. This requires that we understand a little geometry. There are some exercises at the end of the chapter which will enable you to see how to proceed in special cases.

The assumption that the transformations will derive from the action of a Lie group, or some neighbourhood of the identity in a Lie group, is overly optimistic. It is reasonable to suppose that this is largely the case however, and that many transformations can be approximated adequately in this way.

As well as factoring out the transformations under which the trajectory classification is invariant, or at least giving it a shot, there is another desideratum: stability under noise. The measurements of the state of the system at different times will generally be, in this imperfect world, contaminated with noise. This will have an effect on the job of estimating the dimension and structure of any manifold of transformations. It would be as well therefore to smooth the data first. Unfortunately, the smoothing process might easily eliminate the significant parts of the signal or trajectory. For example, in handwriting, cusps and regions of high curvature are usually highly informative, often being separators between different regimes. Having a curve blunted just when it is starting to point somewhere interesting would not be good for recognition. The assumptions behind most smoothing operations, that we have a stationary or at least slowly changing process which is afflicted by gaussian noise, may be grotesquely inappropriate. It may be necessary to divide the trajectory up into segments, each of which is relatively homogeneous, but where there are plainly distinct regimes. If one does this with cursive writing, one gets what might reasonably be called *strokes*, and one can also find such things in the speech signal. I shall discuss this at some length in the chapter on Syntactic Pattern Recognition.

Next Up Previous Contents

Next: Exercises **Up:** Continuous Dynamic Patterns **Previous:** Adaptive Filters, Kalman Filters *Mike Alder*
9/19/1997

Exercises

1.

A time series consists of the numbers 1,0,-1,0,1,0,-1,0, repeated many times. Writing it as a set of points in $\mathbb{R}^2$ we get the vectors:

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \ldots$$

which we can see is the set of iterations of the matrix

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

on an initial vector $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Now suppose some noise is added to this time series, with a zero mean gaussian with variance about 0.1 generating a random number which is added to the original series. This means that iterating from the given (noise free) initial state will produce four clusters in $\mathbb{R}^2$. The fact that there are four tells you that the original time series is periodic with period four, and the fact that each cluster is a spherical gaussian tells you that you have additive gaussian noise which is uncorrelated. Filtering the series can be done by replacing each gaussian cluster with its mean.

Experiment by modifying the example so as to (a) increase the period, (b) take longer time delay vectors, (c) change the noise so that it is MA filtered gaussian white noise. Investigate the kinds of point set which you get and decide how you might filter the noise out.

Experiment by taking some matrix map, some initial point, and some noise process. Iterate the map plus the noise and examine the point set you get. If you choose low dimensions you can easily project the sets onto the screen of a computer and gloat over them. What common sense suggestions do you have for filtering out the noise?

2.

Use the *fview* program (see the bibliography) to obtain samples of the digits zero to nine spoken by, say, five different people. Now get a program to select a sixth speaker, extract the same words and rename the files so as to conceal the words spoken but to store the encoding in case of fights

later. Get all your friends to try to work out from looking at the sample trajectories what words the mystery speaker is speaking. Provide a bottle of wine for the person who gets most right. Get the people who got them right to explain how they did it, and try to write a program to use the same methods.

3.

You are given a set of points in the plane in two categories, Noughts and Crosses. The Noughts are represented by the data points:

$$\begin{pmatrix} -1 \\ 1 \end{pmatrix}, \begin{pmatrix} -0.8 \\ 0.64 \end{pmatrix}, \begin{pmatrix} -0.6 \\ 0.36 \end{pmatrix}, \begin{pmatrix} -0.4 \\ 0.16 \end{pmatrix}, \begin{pmatrix} -0.2 \\ 0.04 \end{pmatrix}, \begin{pmatrix} -0.1 \\ 0.01 \end{pmatrix}$$

$$\begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}, \begin{pmatrix} 0.3 \\ 0.09 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.25 \end{pmatrix}, \begin{pmatrix} 0.7 \\ 0.49 \end{pmatrix}, \begin{pmatrix} 0.9 \\ 0.81 \end{pmatrix}$$

and the Crosses by the data points

$$\begin{pmatrix} -1 \\ 1.5 \end{pmatrix}, \begin{pmatrix} -0.8 \\ 1.14 \end{pmatrix}, \begin{pmatrix} -0.6 \\ 0.86 \end{pmatrix}, \begin{pmatrix} -0.4 \\ 0.66 \end{pmatrix}, \begin{pmatrix} -0.2 \\ 0.54 \end{pmatrix}, \begin{pmatrix} -0.1 \\ 0.51 \end{pmatrix},$$

$$\begin{pmatrix} 0.0 \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0.3 \\ 0.59 \end{pmatrix}, \begin{pmatrix} 0.5 \\ 0.75 \end{pmatrix}, \begin{pmatrix} 0.7 \\ 0.99 \end{pmatrix}, \begin{pmatrix} 0.9 \\ 1.31 \end{pmatrix}$$

First view the data. This should lead you to suspect that using one gaussian for each category would not be a good idea, and you should also rule out the use of a single unit Perceptron. You then have to choose between the hypotheses that the data shows signs of a group action of transformations which might be learnable, and that it doesn't. In order to test this, you take each data set, a radius of about 0.4, select points and compute for all the points within a radius of this resolution the covariance matrix.

You test to see if the results indicate a dimension of one. (Actually, one is the only one worth trying for in the circumstances, so you could give this a miss.) Having found that this is the case, you decide to try to fit a polynomial function to each category.

Compute a low order polynomial to fit each of the categories. This is your first pass at an attempt at obtaining a transform under which the category is invariant. Proceed as follows: linearise the points of one category by taking the polynomial for the Noughts $y = f(x)$ and sending each point $\begin{pmatrix} x \\ y \end{pmatrix}$ to the point $\begin{pmatrix} x \\ y - f(x) \end{pmatrix}$ .Now forget about the first co-ordinate and simply

store each point by its residue. The results should make pattern recognition rather simpler. Test the

legitimacy of using the same polynomial on both data sets by computing the variance of the residues y-$f(x)$ and seeing if the variance is different for each category, or by curve fitting the residue and testing to see how non-linear it is. If the two functions f_0 and f_X for the two categories had been very diffferent, what might you have tried?

4.

Given the simplicity of the last exercise, find an embedding of a k-dimensional cube in $\mathbb{R}^n$ for $k \approx 4$ and $n \approx 12$ which gives, when some values are selected randomly and when small amounts of noise are added, a finite data set lying close to a k-manifold This is going to be a data set obtained by applying a k-dimensional space of transformations to a single point. Now change a few parameters in your embedding to get a different embedding of a different data set. Again generate some more points. Note that I took k=1, n= 4; $x = t$, $y=t^2$; $x =t$, $y=t^2$+0.5 when I did this in the last exercise. I had zero noise.

Now can you get back again to recover the embeddings given the data points? If you can, describe what happens in the space of embeddings as you go from one category to another. Is there any reason for trying to linearise the space so as to be able to factor out the transformation? After all, it would be simplest to decide what category a datum is in by measuring its distance from the best fitting manifold, why should you go to any more trouble than this? (Hint: There may be more problems in the same space coming down the pipeline!) You might consider the low dimensional case where one category has $y = x2$ as a fitting manifold and the other has $y = x^4 + x^2 + 0.3$ as its.

Find a non-linear transform of $\mathbb{R}^2$ which allows you to factor out one component in order to get rid of the transformation group action.

5.

Learning Transformations You are given a set of points of type Nought which lie on the graph of $y = x^2$ and points of type Cross which lie along the graph of $y = x^4 + x^2 + 0.3$. All such points have X-values uniformly between -1 and +1. We shall suppose you have plenty of points so as to be able to fit the curves with high precision. You are also given a point of a third category Square, at location $\begin{pmatrix} 0.5 \\ 0.4 \end{pmatrix}$. Where else might you find points of type Square?

6.

The last three exercises had nothing much dynamic about them, and you might feel that they should have been treated earlier with static recognition. Quite so. Now I ask you to produce some pseudo-speech data corresponding to utterances by a range of speakers.

You take a set of measurements of different speakers saying the vowel /AA/, as in English Cat and Bag, and discover, after binning them into 12 filterbank values that the sounds occupy a region of the space $\mathbb{R}^{12}$ which is approximated by a single gaussian distribution, and has dimension about six, i.e.

the first six eigenvalues are reasonably large, and the last six are all pretty much the same, small,

and can be interpreted as noise. You repeat for seven other vowel sounds and discover that the same is essentially true for each of them, that the centres of the gaussians occupy a plane in the space, and that the gaussians are more or less shifts of each other in this plane. The principal axes of the gaussians all stand out of the plane at an angle of about 30^o, like six dimensional whales leaping out of a flat sea, all about half out and all pointing in the same direction. An impressive sight.

You entertain the hypothesis that the distributions are telling you that there is a six dimensional space of transformations which can be performed on a vowel utterance in order to encode information about how loudly it is spoken, the age and sex of the speaker, and so on.

(a)

How might you test this hypothesis?

(b)

If you were satisfied that the hypothesis is tenable, how would you use the data in order to normalise a new trajectory in the vowel space?

(c)

What about a trajectory representing a word such as `` wish" or ``whiff"?

(d)

Construct some data satisfying the conditions described and examine it carefully, comparing it with real speech data.

7.

Consider the double spiral data set of the last chapter. Although somewhat implausible as a model for some transforms of a basic category, it is plain that there is a certain similarity between the case of an embedding of an interval arising from some family of transformations and the spiral structure of the data sets. In fact one *could* represent the spirals as the embedding of an interval. The question arises, can one extend the arguments used to deal with embeddings of manifolds which are not too far from being flat, to cases such as the double spiral in which there is plainly some kind of structure to the data? It is plain that we could perform the same trick as before: first choose a suitable radius, then compute covariance matrices for points lying inside balls of the radius, use the coincidence of all of them having one eigenvalue much bigger than the other to argue that the data is essentially one dimensional, and then try to fit a curve. It is the last part which will give some troubles to an automatic system; doing it with polynomials does not usually give good extrapolation

properties. Try it and see.

Next Up Previous Contents

Next: Bibliography **Up:** Continuous Dynamic Patterns **Previous:** Fundamentals of dynamic patterns

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Discrete Dynamic Patterns **Up:** Continuous Dynamic Patterns **Previous:** Exercises

Bibliography

1. Peter Ladefoged, *Elements of acoustic phonetics*, Oliver and Boyd 1962.
2. Flanagan, James Loton *Speech analysis, synthesis, and perception*, Springer, 1972.
3. Norman J. Lass.(Ed) *Contemporary issues in experimental phonetics*, Academic Press, 1976.
4. Paul A Lynn, *An Introduction to the Analysis and Processing of Signals*, Macmillan, 1985.
5. Lawrence R.Rabiner, Bernard Gold, *Theory and application of digital signal processing* , Prentice-hall,1975.
6. Lawrence R. Rabiner, Ronald W. Schafer *Digital processing of speech signals* , Prentice-Hall, 1978.
7. Lawrence R. Rabiner *Fundamentals of speech recognition*, Prentice-Hall, 1993.
8. Wayne A. Lea, editor *Trends in speech recognition*, Prentice-Hall, 1980
9. D. R. Reddy, editor, *Speech recognition : invited papers presented at the 1974 IEEE symposium*, Academic Press, 1975.
10. Renato De Mori and Ching Y. Suen (Eds) *New systems and architectures for automatic speech recognition and synthesis*, (Proceedings of the NATO Advanced Study Institute on New Systems and Architectures for Automatic Speech recognition and Synthesis held at Bonas, Gers, France, 2-14 July 1984) Springer-Verlag, 1985.
11. Kai-Fu Lee (Foreword by Raj Reddy) *Automatic speech recognition : the development of the SPHINX system*, Kluwer Academic Publishers, 1989.
12. Y Linde, A Buzo and R.M. Gray *An algorithm for Vector Quantizer Design*, IEEE Trans.Comm. COM-28(1)pp84-95 1980.

13.

Fred Jelinek, *Principles of Lexical Language Modeling for Speech Recognition*, IBM Report: Continuous Speech Recognition Group, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA.

14.

Fred Jelinek *Self-Organized Language Modeling for Speech Recognition*, IBM Report: Continuous Speech Recognition Group, IBM Thomas J. Watson Research Center, Yorktown Heights,NY 10598. USA.

15.

Lalit Bahl, Frederick Jelinek and Robert Mercer *A Maximum Likelihood Approach to Continuous Speech Recognition*, IEEE Trans Paattern analysis and Machine Intelligence Vol PAMI-5 No. 2pp179-190 March 1983.

16.

L.R. Bahl, P.F. Brown, P.V. deSouza, R.L. Mercer and M.A. Picheny *A method for the Construction of Acoustic Markov Models for Words*, IEEE Trans. Speech and Audio Processing Vol.1 No. 4. pp443-452, October 1993.

17.

Robert Linggard *Electronic synthesis of speech*, Cambridge University Press, 1985.

18.

Gareth Lee *fview*, ftp site: ciips.ee.uwa.edu.au , IP no. 130.95.72.69, in directory pub/fview. 1993.

19.

S. M. Bozic *Digital and Kalman filtering : an introduction to discrete-time filtering and optimum linear estimation*, Wiley, 1980.

20.

T. Kailath *Lectures on Wiener and Kalman filtering*, Springer-Verlag, c1981.

21.

Karl Brammer and Gerhard Siffling *Kalman-Bucy filters* , Artech House, c1989.

22.

Walter Vandaele *Applied time series and Box-Jenkins models*, Academic Press, c1983.

23.

Norbert Wiener *Extrapolation, interpolation, and smoothing of stationary time series : with engineering applications*, M.I.T. Press, 1949.

24.

George E. P. Box and Gwilym M. Jenkins *Time series analysis : forecasting and control* , Holden-Day, 1976.

25.

Maurice G. Kendall *Time-series* , Griffin, 1973.

26.

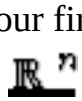
Joseph L. Doob *Stochastic processes*, Wiley, [1953].

Discrete Dynamic Patterns

Consider, first, the spelling checker. It consists of a large dictionary of words in a particular language, and it is useful for compensating for the deficiencies of a progressive education, since if you have written a page of text the chances of a spelling error are quite high, and we all wish to conceal our intellectual inadequacies. It also has use when trying to do OCR, since if we are uncertain as to whether a given character is an `s' or a `5', we can look at the surrounding letters and use the fact, for example, that `Profes5or' is not a word but `Professor' is. When we do this we are using syntactic information to resolve an uncertainty. We are *smoothing* the word, filtering the string of characters, somewhat as we might use a moving average filter to use the neighbouring values in a time series to modify a particularly egregious value. It is plain to everyone who has tried to decipher bad handwriting that such methods are applied a good deal in human information processing.

We shall be concerned in this chapter with long strings of symbols, short strings of symbols, and what are known as *formal languages*, which is to say sets of strings of symbols. We do not insist that the language arise from people speaking or writing, although these do give particular cases, but we examine the subject in vastly more generality. We shall also be concerned with *stochastic languages*, briefly mentioned in the last chapter, where there is, for each string in the language, some number saying how often it comes up.

The reflective reader will, of course, be wondering why we devote a whole chapter to such matters.

Having focussed to a large extent of the recognition of characters in discussing applications, the reader *might* think that this is the important connection with Pattern Recognition, since strings of characters form words, strings of words form sentences and sentences are (or aren't) grammatical. Anyway, there is presumably *some* connection between formal language and natural language, and characters come into natural language. Thus the reader seeking for connections between things, that is to say the reader who is thinking about what he reads, might hazard a guess that we are going to be concerned with using a spelling checker to improve the chances of a correct reading of some characters. He might also guess that there is a connection with Speech Recognition, since our first move there, when discussing the standard approach, was to reduce the continuous trajectories in  to trajectories in a discrete space, that is to say, symbol strings. So maybe we are going to enlarge our perspectives on Speech Recognition on the side.

It is true that one aim of this chapter is to show how to extract information about which strings of symbols occur and which don't in a page of text so as to improve recognition of characters, but this is the tip of a rather humongous iceberg. The problem in the case of strings of symbols is fairly easy to understand, but the ideas go across to cases which are on the face of things very different. For example, I shall explain in the next chapter how to clean up an image by using knowledge drawn from other images. This is a clever trick when human beings do it, and is known to psychologists as *Transfer of Learning*,

and making a program do it is even cleverer. It is essentially a rather considerable generalisation of using a dictionary to resolve uncertainty or correct a preliminary decision at a lower level, as in `Profes5or'. The generalisation is so extreme as to require a good deal of preliminary technicality however, and this is one purpose of the present chapter.

We observe that our low level objects, characters in the above example, are combined together to form higher level objects, words in the example. You would not be the first person to notice that it doesn't stop there; words are aggregated into phrases, phrases into sentences, sentences into paragraphs, and so on. There are hierarchies out there. The idea of a `high level object' is rather an interesting idea to try to make precise. Cast your mind back to when you learnt to drive a car. As a rank tiro, you may have had to think quite hard when the instructor gave the order `turn left'. It requires slowing down, i.e. taking your right foot up a bit, signalling left, looking in the mirror to ensure that the driver of the car behind knows you are slowing, disengaging the gears by depressing the clutch (push your left foot down), and moving the gear lever from where it is to somewhere else (where, for God's sake?). In addition you have to keep an eye open to ensure that the turn is not prohibited by some road signs and that you won't run over any little old ladies. Oh, and you'd better remember to turn the steering wheel as well. Seen from this point of view, turning left is a pretty complicated business. Actually, the muscle contractions required to depress the clutch and reduce the pressure on the accelerator are also pretty complicated and you took years to learn to do them; most of the time between birth and three years of age in fact. And now you plan a drive of a hundred miles without, I hope, a trace of panic. It is clear that there is a chunking of low level actions consisting of muscle fibres firing, into larger things like lifting a foot, chunking of actions like lifting feet, pushing legs, turning the wheel and so on into `turning left', and a further chunking of turns and stops and starts in order to get you from A to B. Although why anybody should want to get to B is far from clear to anybody who, like myself, has never visited A.

Now the intuitive notion of chunking low level things into higher level things, and so on, and the corresponding decomposition of some things into sub-things occurs all over the place. If you are obliged to write a computer program for example, then you are asked to write a string of symbols which defines a map. The program will have some input, possibly several different sorts, and some output. You therefore are faced with the problem of constructing this map using the more primitive maps given in the language such as `+', `*', `&', `if' and so on. You also have standard ways to glue these primitive maps together. It is good programming practice to write your program as a set of procedures, and a procedure is a sub-map. A well written program is likely to have elements of a hierarchical structure to it, in other words it is a chunk of chunks of... of the given primitives. This is more obvious in the so called applicative or functional languages (the French word for map is *application*), but it holds in all of them. So again, chunking occurs, and the rather woolly idea of `high level' objects and lower level objects makes sense. And as observed, chunking occurs in natural language. In the case of a printed book, we may consider at the lowest level the pixels of ink on paper, then the edges which are defined by the pixels, then the strokes made up of edges, then the characters built up from strokes, then the words built up from characters, the phrases built up from words, the sentences from phrases, the paragraphs from sentences, the chapters from paragraphs, and finally the book from chapters. And you could, in good conscience, insert some intermediate levels if you wished.

The intuitive notions are quite compelling and are even well known to psychologists, but the Mathematician is impelled to ask how to define the terms properly. Talking about things with no very clear meaning does very well in the Soggy Sciences, but it doesn't cut the mustard in real science or in engineering, and I have no wish to be mistaken for a psychologist as the term is currently widely

understood, anymore than anyone in a Chemistry department might wish to be taken for an alchemist. Nevertheless, it has to be admitted that what we are studying was identified by psychologists first as an important aspect of cognitive processing in human beings. Our job will be to make the ideas precise. This will involve us in defining syntactic structures of a continuous sort, and it would be a good idea to get the standard ideas clarified in our minds first. The really interesting parts of this chapter therefore will be the sections that have to do with being able to define clearly what is meant by a chunk, for these have extensive applications to all sorts of pattern recognition. If you want to describe the shape of a set of points in it may be convenient to find a description of parts of the set and the relationship between the parts, and here it pays to know what you mean by the terms.

It is important to clarify ideas about structured objects with substructures, and the case of a sequential or temporal relation between the substructures is the easiest to deal with, so it makes sense to tackle this case first. This chapter is therefore a bridge between classical pattern recognition and the modern syntactic approach.

In the hope, then, that your interest has been aroused enough to carry you through the forest of definitions which are coming up, we proceed to investigate formal languages and streams of symbols.

- Alphabets, Languages and Grammars
  - Definitions and Examples
  - ReWrite Grammars
  - Grammatical Inference
  - Inference of ReWrite grammars
- Streams, predictors and smoothers
- Chunking by Entropy
- Stochastic Equivalence
- Quasi-Linguistic Streams
- Graphs and Diagram Grammars
- Exercises
- Bibliography

Next Up Previous Contents

Next: Alphabets, Languages and Grammars **Up:** An Introduction to Pattern **Previous:** Bibliography

Mike Alder
9/19/1997

Alphabets, Languages and Grammars

- Definitions and Examples
- ReWrite Grammars
- Grammatical Inference
- Inference of ReWrite grammars

Mike Alder
9/19/1997

Next Up Previous Contents

Next: ReWrite Grammars **Up:** Alphabets, Languages and Grammars **Previous:** Alphabets, Languages and Grammars

Definitions and Examples

Definition

An *alphabet* is a set of objects called *symbols*.

Alphabets are usually finite, but I shall not so limit myself. Think of the ASCII character set to make your ideas concrete, but watch out for more bizarre examples coming up.

Definition

A *string* on an alphabet is a finite sequence of symbols from the alphabet.

Definition

A *language* over an alphabet A is a set of strings on A.

Languages may be finite, but these aren't the interesting ones. It is usual to define A* to be the set of all possible strings over A, which makes A* a language. This allows us to define a *language over A* as a subset of A*.

Example

If A is the set of ASCII characters, the set of correctly spelled English words is a language. The set of syntactically correct PASCAL programs is another. The set of English sentences is a third.

Example

If A is the set of words in the Oxford English Dictionary, then the set of English sentences is a language over A. It is a different language from the same set of sentences over the ASCII character set.

Definition

A *recognition grammar for a language* $\mathcal{L}$ *over A* is an algorithm which when given a string on A will output either a 1 or a 0, and will output a 1 whenever the string is in $\mathcal{L}$ and a 0 when it isn't.

I shall not define an *algorithm* here, you can think of a computer program of some sort if you want to make it reasonably definite.

The definition I give is not quite standard, because the usual next move in the literature is to go to some kinds of restrictions on the algorithms which might be employed and I do not wish to consider the usual class of restrictions. I shall discuss this further below.

Definition

A *generative grammar* for a language $\mathcal{L}$ over A is an algorithm which has no input but produces strings on A such that only strings in $\mathcal{L}$ are produced and every string is produced eventually.

If the alphabet is finite, A* is countable, so generative grammars may exist. Indeed it is hard at first glance to see how they might not, since how can I tell you what $\mathcal{L}$ *is*, except by specifying a generative grammar? You may wish to brood upon this point.

Definition

A *stochastic language over A* is a language over A together with a real number in [0,1], *p*(*w*), assigned to each string *w*, so that (a) if*w* < *w*' (*w* is a substring of*w*') then $p(w') \leq p(w)$ and (b) the limit of the sums of these numbers over the entire language exists and is 1.

Example

Let A consist of the set a,b and $\mathcal{L}$ be the set

$$\{aba, abba, \ldots ab^n\ a, ..\}$$

for every positive integer *n*. Let the number $p(ab^na)$ assigned to the string ab^na be $\frac{1}{2^n}$.

Example

Let A be the ASCII character set, $\mathcal{L}$ the set of English words in the Oxford English dictionary, and for $w \in \mathcal{L}, p(w)$ is defined to be the relative frequency of occurence of the word in the present book.

Definition

A *stochastic recognition grammar* for a stochastic language $\mathcal{L}$ over A is an algorithm which when given a string *w* on A returns *p*(*w*).

Definition

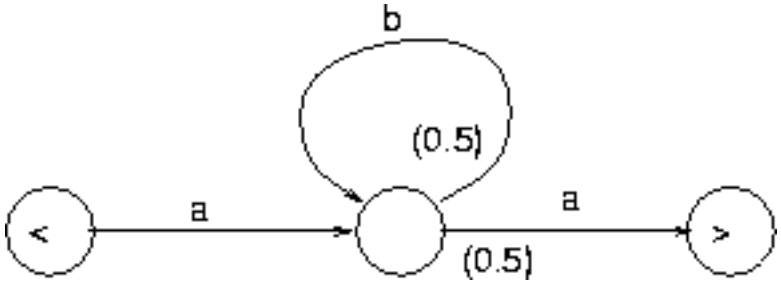
A stochastic generative grammar for a stochastic language $\mathcal{L}$ over A is an algorithm with no input which produces strings on A so that the relative frequency of the string *w* in a sample of size N strings produced by the algorithm, tends to p(w) as N gets larger.

Example

Suppose we are given the finite state diagram of *fig*.7.1 and a pseudo-random number generator. The algorithm traces a path through the diagram from < to > in order to generate a string, and whenever it makes a jump from one state to another, or back to itself via a self loop, it produces the symbol attached to the transition arc. As soon as it gets to > it starts over again to get a new string. The algorithm decides

whether to go by one arc or another out of a state by running its random number generator and deciding in accord with the probabilities marked on the arcs out of a state. Thus if there is only one arc out of a node or state it moves along the single arc, while if there are two with equal probabilities 0.5 as in the diagram, it does the pseudo-random number equivalent of tossing a coin. It is easy to persuade oneself that it produces only strings $ab^n a$ for positive integer n, and that the frequencies of these strings will tend to $p(ab^n a) = \frac{1}{2^n}$, given only moderate optimism about the pseudo-random number generator's capacity to reliably simulate a 0.5 probability choice.

Figure 7.1: A Markov Process for generating a stochastic language.



Definition

A finite network of nodes and arcs, with probabilities attached and symbols attached to the arcs, such as that illustrated in the last example, is known as a *first order markov process* for generating the stochastic language. If we take the probabilities away and forget about frequency counts, the diagram is known as a *finite state grammar* or *regular automaton* for the language.

Clearly, some languages can have finite state generative grammars for them and others may not have such grammars. The set of *finite state languages*, otherwise known in some quarters as the *regular languages* are those for which a finite state grammar can be found. The set is widely regarded as an important subset of all possible languages, and deciding whether a language is regular or not is a problem of some interest. It is trivial that all finite languages are finite state, but the converse is obviously false.

The distinction between generative grammars and recognition grammars can be thought to be important or regarded as an inadequacy of the language and a sign of a poorly constructed framework for talking about things. The distinction is nugatory for stochastic grammars for languages over a finite alphabet. If I have a recognition grammar for such a language, then I merely need to produce all the strings on the alphabet in some order and then test to see if the string is in the language. If it is, I put it in the box $\mathcal{L}$ and I have generated a string of the language. This clearly works for the stochastic and non-stochastic cases. Conversely, if I have a generative grammar and you give me a string w and want to know if it is in $\mathcal{L}$, I need to generate the strings until I can match the given string or fail to match it. But a failure to match it in any finite number of moves is inconclusive for the non-stochastic case. Whereas in the stochastic case I can obtain estimates for $p(w)$ by generating sequences of strings and counting

occurrences of *w*.

Of course, one might doubt whether this is an algorithm for assigning a probability or relative frequency of occurrence to a string; it only yields estimates based on finite samples. A platonist philosopher or even a logician might argue that only production of the whole infinite set would yield the procedure, and this cannot be accomplished by a real program running on a real machine. Much depends here on what one accepts as an algorithm, and also on what one regards as an algorithm producing a real number measuring a relative frequency of occurrence. I apply a commonsense test of meaning here. For my purposes, a program which assures me that the probability of an event is 0.3333 is (a) feasible to write and (b) may be useful to run. Likewise, being told that the probability of occurrence of a string is less than 0.01 is helpful, whereas being told that I haven't got a stochastic recognition grammar unless presentation of a string of symbols yields an infinite decimal string, specifying the probability, would be bad for my blood pressure. My choices are either to specify what I mean very carefully indeed, as I should have to if I were writing a program for a computer or a text for Pure Mathematicians, or rely on the maturity and common sense of the reader to work out what I intend for practical purposes, which is quicker but a little risky. I choose to take a chance. I may regret this later.

By the *trace* of an algorithm I mean the sequence of operations it performs on a particular input.

Definition

The trace of a recognition grammar for a particular string is called a *parse* of the string.

Next Up Previous Contents

Next: ReWrite Grammars **Up:** Alphabets, Languages and Grammars **Previous:** Alphabets, Languages and Grammars *Mike Alder*
9/19/1997

ReWrite Grammars

Noam Chomsky is a conspiracy theorist who used to be a theoretical linguist; the Chomsky Hierarchy refers to language types rather than degrees of authority. Chomsky's work in formal language theory has been described by some as `seminal' .

At the bottom of the ladder sit the finite state languages. If we want to we can think of them as being the languages we get from hopping about a finite state diagram, or we can cast them into ReWrite form as in the following example:

The language $\{ab^n a : n \in \mathbb{N}\}$ can be represented by the finite state diagram of *fig.* 7.1 it can be expressed rather more elliptically as

$$\{aa, aba, abba, abbba, abbbba, \cdots\}$$

Now we use three alphabets to describe it, and ReWrite rules. The first alphabet has only one symbol in it, *S* (which some say stands for `sentence', and others for `start'), and you are allowed to ReWrite it to the string *ABA* on a quite different alphabet containing two symbols, *A*,*B*. We write this in the form:

$$S \longrightarrow ABA$$

The symbols *A* and *B* can be rewritten into the last alphabet, the symbols of the alphabet of the language for which we are trying to obtain a grammar, the alphabet $\{a, b\}$. The allowed ways of doing this, together with the above ReWrite for completeness sake, are:

It is easy to see that if you simply follow the rules which are to start with an S and rewrite the strings you get by rewriting the symbols in them and stop only when there is nothing more to do, then you must generate one of the strings of the given language. Also, every string of the language can be obtained in this way.

It is also easy to see that you could make this into a stochastic grammar for the stochastic language generated with the probabilities given by *fig*.7.1 by putting probabilities on which rule you choose when there is a choice of rewriting a particular symbol. This is a somewhat unusual way of representing a Markov Model, but it generalises to other languages which are not describable as Markov Models.

With only a little thought, you can convince yourself that any finite state diagram can be turned into a ReWrite grammar. The idea is simply to label the nodes of the diagram with letters from a new alphabet,

and write

$$S \longrightarrow xY$$

whenever you can get from the start of the diagram to the node Y by emmitting the symbol x, and then to continue similarly for all the other nodes. We may conveniently use a blank for the symbol for the terminating node or nodes if we have such things.

Conversely, if the ReWrite rules all look like

$$X \longrightarrow xY$$

where X and Y may be the same, and Y may empty, when it represents the terminating node of the diagram, then it is easy to build a finite state diagram of nodes and arrows which produces the same output. Clearly, some compression of this description is allowable; we didn't really need the A symbol in the example at all, and x may be a string of terminal symbols, that is, symbols of the alphabet of the language we actually care about.

In general we may define a ReWrite grammar as follows:

Definition

A *ReWrite Grammar* for a language $\mathcal{L}$ is a finite sequence of alphabets of distinct symbols where the last alphabet is the alphabet of the language $\mathcal{L}$, together with a finite set of *productions* or *rules* of the form

$$P \longrightarrow Q$$

where P and Q are strings of symbols from any of the alphabets. A *derivation* of a string ℓ of $\mathcal{L}$ is a sequence of productions and a sequence of strings, such that the first element of the sequence of strings is a single symbol from the first alphabet, the last string is precisely ℓ, and the move from one string to its succesor is sanctioned by the corresponding rewrite rule: the k^{th} string contains a substring *P* where the $(k+1)^{th}$ string contains a substring *Q*, and the left hand side of the k^{th} production also contains *P* where the right hand side contains *Q*, and where the left hand side of the production is a substring of the k^{th} string.

The alphabets other than the last are called the *intermediate alphabets* and any symbol from them is called an *intermediate symbol*. The last alphabet is called the alphabet of *terminal symbols*, the first alphabet the alphabet of *initial symbols*. Then we require that every string of the language can be produced by a derivation, and no strings not in the language can be so derived.

This is a little untidy: it suffices to have only one initial symbol, by introducing an eaven earlier alphabet if necessary, together with the obvious productions to get from my new initial symbol to your old initial symbols. Similarly, we can get by with amalgamating all the intermediate alphabets into just one. Why bother to make the distinction? Ah, well, there may be reasons coming along shortly.

One of the first restrictions which we could make on the ReWrite Grammars defined above is to stipulate

that you can only go *down* the sequence of alphabets, that is to say, when we have a production

$$P \longrightarrow Q$$

Q is not allowed to contain symbols from earlier alphabets than P. If you reflect on natural language, where you can ReWrite the *S* symbol as, perhaps, $NP * VP * NP$ where *NP* is a new symbol (standing for Noun Phrase) and the sequence of ReWrites which can go on to generate a sequence of ASCII characters constituting a sentence in the English language, you will perhaps find this stipulation attractive. It gives some kind of hierarchical structure to the system.

The most extreme restriction of this kind we might make is that the left hand side of each production is allowed to consist of a single symbol at any level, and the right hand side of a string of symbols at the level one lower. This gives only finite languages such as English. It doesn't do a whole lot for English either. Even a superficial familiarity with the structure of English or any other natural language leaves one feeling that grammars such as this don't get us too far.

An alternative restriction which might be imposed would be to allow the right hand side of a production to contain a string of symbols from the next alphabet lower down followed by an optional symbol from the same level as the left hand side, and to stipulate still only single symbols on the left. This gives us, by the above argument, the finite state languages. This is a weaker restriction, so we get a larger class of languages which can be obtained by this class of ReWrite rules.

To weaken things even further, suppose that you ensure that the left hand side is a single symbol, from some level alphabet, and the right hand side is some mix of symbols from the same level or the next lower level alphabet. I shall call such a grammar a *Context Free* (ReWrite) grammar. If a language may be obtained from such a grammar but not from a finite state grammar, it is called a *context free language*.

As an example, consider the language $\mathcal{L}_{cf}$ defined by

$$\mathcal{L}_{cf} = \{a^n b a^n : n \in \mathbb{N}\}$$

Now it is not too hard to see that this is *not* a finite state language, that is to say there is no finite diagram such that running round it will generate precisely this set of strings.

To convince yourself of this, observe that if you run around a finite state diagram which has no loops in it, then there is some longest string which can be obtained, and all other strings are the same length or shorter. So there can only be a finite number of them. It follows that if the language has an infinite number of strings and is finite state, then the finite state diagram which generates the language must have at least one loop. Extending the argument a little, given such a diagram, there is a longest path (at least one) which does not pass around a loop, and any longer path must go around some loop at least once. So if you have an infinite language which is regular, there is some string in it with the property that generating the string involved going at least once around a loop, so that some of the string represents going around the loop, the bit before (if any) represents getting to the start of the loop, and the part after was obtained by going home after completing the loop. Now if you went around once, you could have gone around twice, three times or any other number of times. Or to say it in algebra, there is for any finite

state language which is infinite, a string ℓ in that language having a substring *p* in it, so that $\ell = apb$ for other strings *a*,*b* (possibly empty) and such that ap^nb is also always in the language, for any $n \in \mathbb{Z}^+$. This result is called the *pumping lemma*, posibly because you get a heart attack if you run around the same loop often enough.

Now apply it to the language $\mathcal{L}_{cf}$. There is, if the language is finite state, some (possibly longish) string in it and a substring ℓ representing an arbitrarily repeatable loop. If this substring ℓ contains the *b*, then there would have to be strings which contain lots of *b*s in the language, and there aren't. So it must consist entirely of *a*s on the left of the *b*, or some string of *a*s on the right of the *b*. Either way this leads to strings a^nba^m for $n \neq m$, which do not in fact occur. So the language is not finite state.

It can however easily be obtained by the following ReWrite system

among others. This is evidently context-free.

Context Free languages can also be described by graphical means; the diagrams between pp 116 and 118 in the *PASCAL User Manual and Report* by Kathleen Jensen and Niklaus Wirth, Springer Verlag, 1975 follow a Backus-Naur formalism (Rewrite Grammar) description of the syntax of the programming language PASCAL. It is easy to see that we could put all the labelled boxes together into one big diagram. They do not, however, constitute an automaton or finite state diagram (or finite state machine) because a subdiagram consisting of a box called *factor* can contain inside itself a sequence consisting of a term `not' followed by the box *factor*. This allows for recursion. The PASCAL diagrams have symbols associated with nodes rather than arcs, but it is easy to persuade oneself that this is a minor difference of convention, and that if I have a finite state diagram where the symbols are emitted at nodes, you can easily construct one in which symbols are emitted along arcs so that both systems produce the same strings, and *vice versa*. Similarly with the diagrams for PASCAL, which determine what is called a *Push Down Stack Automaton*. To represent such a machine we may need to have diagrams containing little boxes with a suitable symbolic label so that a path can take one into and thence out of each such box, and to also have a separate diagram telling you about what is inside the box. What may be inside the box is a path from input to output which passes through other such boxes or even the same box again. It is not too hard to prove that such automata can generate and recognise any Context Free Language and that all languages generated and recognised (or *accepted*) by such automata are context free. This allows for an extension of the pumping lemma for CF languages; the proofs of all these claims are left to the reader to work out, or he can cheat by examining the standard books on the subject. Working out the proper definitions of the terms is probably the only bit that is much fun, and it would be wrong of me to take this pleasure from him.

The next level of generality in the Chomsky Hierarchy was to simply ensure that the right hand side of any production had to contain at least as many terminal symbols as the left hand side. Such grammars are called *Context Sensitive*, as are the languages which can only be obtained from such grammars. (It is easy to construct a context sensitive grammar for a finite state language, merely unnecessary).

And finally, the Unrestricted ReWrite Systems allow anything whatever. It is a fact that any language which can be generated by a Turing Machine can be generated by some ReWrite system, and *vice versa*, and Church's Thesis is the claim that anything that is effectively computable can be computed by a Turing Machine. So if you are willing to buy Church's Thesis, any algorithm whatever for generating strings can be realised by a ReWrite system if you feel so inclined. This explains why I use the term `algorithm' in the definitions at the beginning of the chapter.

Computer Scientists are familiar with the Backus-Naur Formalism, much used for specifying the structure of Computer Languages, as a slightly disguised version of the Rewrite form of a grammar; the Metamathematicians got to it first. The principal applications of formal language theory of this sort are (a) in designing Compilers, where there is virtually no application of a body of theory but some value in the discipline of thought forced by the terminology, and (b) keeping a certain sort of Pure Mathematician off the streets.

Next Up Previous Contents

Next: Grammatical Inference **Up:** Alphabets, Languages and Grammars **Previous:** Definitions and Examples *Mike Alder*
9/19/1997

Grammatical Inference

It is a very nice thing to have a grammar for a language, and represents a nice compact representation of the language. It is a *model* for the language in a sense which you should recognise as congenial to a follower of Rissanen. The question is, how are such things to be obtained?

Definition

The *Grammatical Inference Problem* is, given a language $\mathcal{L}$ over an alphabet A, to obtain a grammar for it. This holds for both stochastic and non-stochastic languages; for a stochastic language one seeks a stochastic grammar.

In real life, one is actually given a sample of strings produced by some process and the sample is finite. The real life problem is to model the process economically. Thus the next example is a sensible problem which can be made absurd with no trouble at all:

Example

You are given what are claimed to be 15 strings produced by a stochastic process of an unknown sort. The strings are *aba* (8), *abba* (4), *abbba* (2) and *abbbba* (1), where the numbers in parentheses give the counts of the string. You are required to model the process and use your model to obtain an estimate of the probability of seeing *abbbbbba*.

Without some extra constraints such as a class of models from which to pick, the problem has no unique solution. For example, you might model the data by saying these are the only strings there are. This is just the SureThing or Predestination model again. Your estimate of the probability of getting *abbbbbba* is that it is zero. This is very incautious of you, since if it occurs your model will get an infinite amount of `surprise', the shock of which might disable you. On the other hand, you might assign a probability of $\frac{1}{2^n}$ for getting ab^na, and of zero for getting any other string. This would also be incautious, since if the next string were *abab* you would get infinite surprise. On the other hand, my instincts and intuitions tell me that I would sooner chance getting *abab* than *abbbbbba*, which looks to me much more like *abbbba*, which I have seen.

This is telling us that my own personal grammatical inference system has more similarities to a model with $p(ab^na) = \frac{1}{2^n}$ for *n* a positive integer than it does to the finite model with $p(aba) = \frac{8}{15}$, $p(abba) = \frac{4}{15}$, $p(abbba) = \frac{2}{15}$, and $p(abba) = \frac{1}{15}$. And I *do* have some sort of personal grammatical inference system, because I do have different expectations about getting other strings from the same source as produced the 15 samples. The question of what kind of expectations I in fact have, and how I obtain them from contemplating the strings would appear to be a study of my personal psychology; not, on the face of things, a very interesting object of thought. If it should turn out that your expectations are very similar to mine and may have been obtained in the same way, then the subject immediately becomes more interesting. And if it should turn out that the kinds of expectations about what can occur in future for sequences of events of a kind such as might interest a cockroach are also obtained by a similar process, then that process becomes very interesting indeed. Now even a cockroach experiences a sequence of things happening to it, and has an interest in predicting the next one or two. Cockroaches, in other words, have to perform grammatical inference too, just like you and me. It seems possible then that we might all use similar methods, which means that in examining these methods we should be saying something about how brains use data to obtain expectations of what might happen next. This hypothesis might seem wildly implausible, but it is (a) economical and

(b) susceptible to some kind of analysis and experimentation. It is therefore defensible as a first attempt at making sense of the phenomenon of learning as carried out by brains.

Definition

A *local (stochastic) grammar of order k* for a language $\mathcal{L}$ is a set of *k*grams, i.e. strings of length *k*, all of which are substrings of strings in $\mathcal{L}$, and such that every substring of length *k* of a string of $\mathcal{L}$ is in the set of *k*grams. If the language is stochastic, each *k*gram is accompanied by its relative frequency in the language among those *k*grams having the first *k*-1 symbols the same. For strings of length less than *k*, the whole string is stored, with its relative frequency in the stochastic case.

The rule for generating strings from the grammar is to choose left segments agreeing with the preceding *k*-1 symbols and to multiply the corresponding relative frequencies conditional on the preceding *k*-1gram. Recognising strings is done in the same way.

Example

Let $\mathcal{L}$ be the stochastic language $\{ab^n a(\frac{1}{2^n}) : n \in \mathbb{Z}^+\}$ and take k to be 3. I write < for the start of a string

and > for its end. Then I get 3grams:

$< ab,\ aba,\ ba >$ from the string *aba* which occurs half the time in any sufficiently large sample from $\mathcal{L}$. I

also get:

$< ab,\ abb,\ bba,\ ba >$ from the string *abba* which occurs in about one quarter of all sufficiently large samples

from $\mathcal{L}$. Given a sufficently large sample or alternatively by using a little arithmetic, we conclude that the local stochastic grammar of order 3 for the language will be:

$$< ab : (1.0),\ aba : (0.5),\ abb : (0.5),\ bba : (0.5),\ bbb : (0.5),\ ba >: (1.0)$$

Now this is a generative grammar and a recognition grammar with or without the frequencies. If we wish to obtain the language with the frequencies, we start off at the begin symbol and have no choice about what comes next, it has to be *ab*, so we write down <*ab*. Now with *ab* as the given 2gram we observe that we have an 0.5 chance of getting an *a* following and an 0.5 chance of getting a *b*, So half the time we go to <*aba* and the other half we go to <*abb*. If we take the first case, we see that we have no choice but to terminate with <*aba*> as our string, which therefore happens half the time. In the second case, we have <*abb* and there is a probability of 0.5 that we follow with an *a*, in which case we terminate with one quarter of our cases being <*abba*>, and an 0.5 probability of following with another *b* to get <*abbb* so far.

It is clear that this will generate the same language as the finite state diagram of *fig*.7.1, so we have an alternative class of grammars to the finite state grammars. If we ignore the probabilities, we get the non-stochastic language back. Finite samples will consistently tend to underestimate *bbb* against *bba* and *abb* against *aba*, but the discrepancies get small fast with sample size.

To recognise a string or to compute its probability is done as follows. Suppose we are given the string *abbba*. We can go immediately to <*ab* and match the first three symbols of the string and write down a probability of 1.0. next we match *abb* and assign a probability of 0.5. Next we match *bbb* with probability 0.5,then *bba* with probability 0.5, and finally we match *ba*> with probability 1.0. Multiplying the five numbers together gives 0.125 as the probability of the string. For the non-stochastic case we simply note that matches to $ab^n a$ are all possible and no others are.

The local grammar, stochastic or not, comes with an inference procedure which is very simple and easy to compute. It

assigns a small but non zero probability to getting *abbbbbba* on the basis of the sample of fifteen strings discussed above, but a zero probability to *abab*, for $k > 1$, which accords tolerably well with my personal prejudices in the matter. For k = 1 we obtain simply the relative frequencies of the symbols, and any string is possible provided that all symbols in the alphabet are included in the sample. It is possible to accomplish grammatical inference for finite state machines with rather more work, and for more general categories of language with more difficulty. The language consisting of $\{a^n b a^n : n \in \mathbb{Z}^+\}$ recall, cannot be described by any finite state machine, although it is easy for a push-down stack machine. The distinctions are dealt with lucidly by Eilenberg, see the bibliography.

It is instructive to infer a local grammar of order *k* for different *k* from the language $a^n b a^n; n \in \mathbb{Z}^+$, or some finite sample of it. The stochastic language $a^n b a^n, \left(\frac{1}{2^n}\right)$ likewise. For $k \approx 20$ and around a million samples, you are unlikely to be able to tell the difference between the language generated by the grammar and the stochastic language. This has some implications for doing practical inference from finite samples. Note that the language generated by the local grammar of order *k* from the whole language as a `sample' is in general a superset of the original language. When it isn't, we say the language is *local of order k*, in this case they are the same. Formally:

Definition

A (stochastic) language is *local of order k* iff it is identical to the language obtained from the local grammar of order *k*.

It is easy to see that the language $\{a^n b a^n : n \in \mathbb{N}\}$ is not local of order *k* for any *k*.

It is clear that the whole business of modelling languages, or more realistically language samples, can be done using a variety of possible classes of model, and that the class of local grammars of order *k* for different *k* at least makes the job easy if not always altogether convincing.

The disenchantment with local grammars of order *k* may be reduced slightly by going to a local grammar of order *k*+1.

Definition

The *Shannon filtration* for a language $\mathcal{L}$ is the sequence of local grammars of order *k* for all positive integers *k*.

We can take the Shannon Filtration of a natural language sample, at least up to the point where *k* is as big as the sample, when it gets pretty silly. It gets pretty silly long before that as a rule; also impractical to store. For instance we may do what IBM's Yorktown Heights Speech Group does and store trigrams of words extracted from English text, together with the counts of the last word in the trigram for all possible bigrams. Shannon used almost exclusively the filtration I have named after him for getting better and better approximations to English. Convergence is not very fast and there are better filtrations. Finding a good one has importance in the study of natural language and such applications as speech recognition and OCR.

Next Up Previous Contents

Next: Inference of ReWrite grammars **Up:** Alphabets, Languages and Grammars **Previous:** ReWrite Grammars *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Streams, predictors and smoothers **Up:** Alphabets, Languages and Grammars **Previous:** Grammatical Inference

Inference of ReWrite grammars

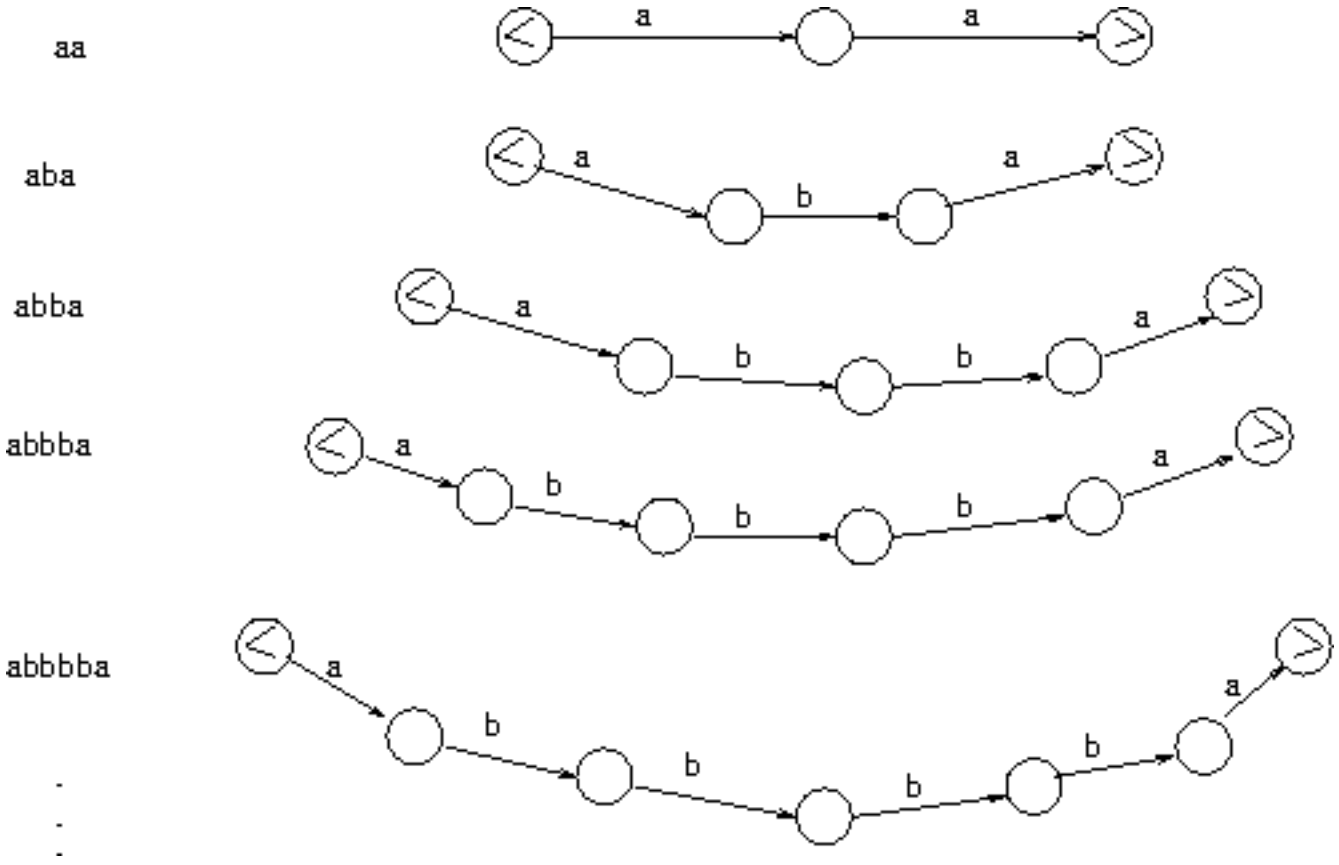
If I specify a language, the question arises, can I infer a grammar for it? For the regular or finite state languages, in the stochastic and non-stochastic cases, the answer is yes. It follows that if I have a finite sample of a (stochastic) language, I can always obtain a grammar, although this may not be the one I want. It is worth a little detour into regular languages in order to examine this matter.

The treatment I shall give extracts the juice from the papers of King Sun Fu on the topic, and makes it much plainer what is going on.

The approach is one of compression. It is easiest to illustrate the method with an example, then to contemplate the abstract issues and finally to formulate the rules. This is how algorithms and theorems are devised.

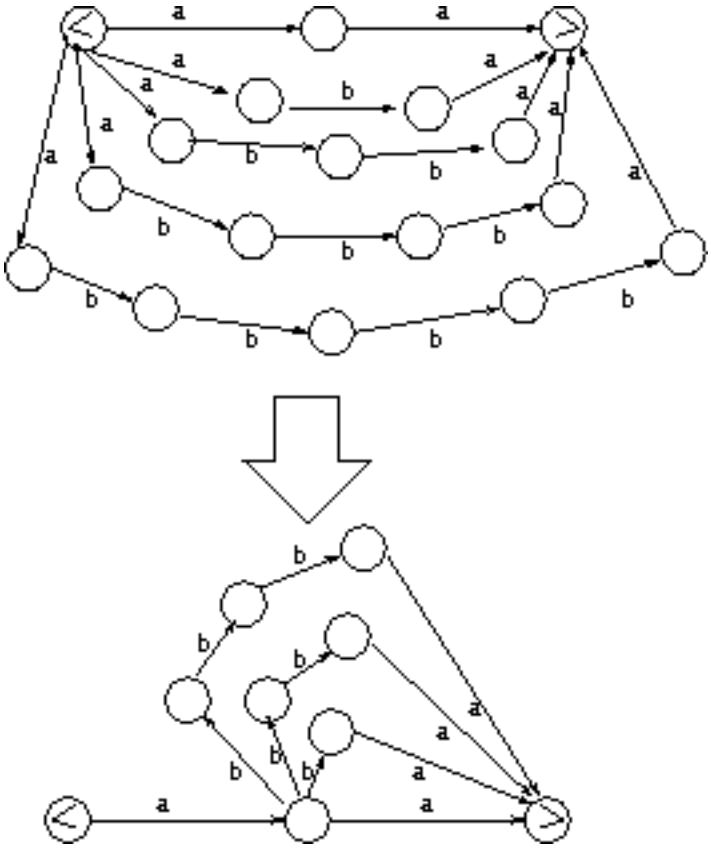
Suppose we are given the language $\mathcal{L} = \{ab^n a : n \in \mathbb{N}\}$, or if you prefer $\{aa, aba, abba, abbba, abbbba, \cdots\}$. We write the lot out as a sort of infinite finite state machine which just happens to be disconnected: any subset of $ab^n a$ for $n \leq N$ will be a perfectly good finite state diagram. I use the symbol < for the begin states and > for the end states. This gives us *fig*. 7.2.

Figure 7.2: A finite state language and an infinite state grammar for it.



The next step is to start gluing together the nodes and arcs, starting with the begin and end symbols, and then proceeding from the left. The next stage is shown in *Fig*.7.3.

Figure 7.3: Gluing bits together.



This gives us the possibility of gluing the first nodes you come to from the start together, and identifying all the arcs also. This is safe because all the arcs have the same symbol, *a*, on them. This gives the diagram of *fig*.7.4, where we see that we can proceed to identify from the left to get the `tower' of *b* symbols.

Figure 7.4: Gluing more bits together.

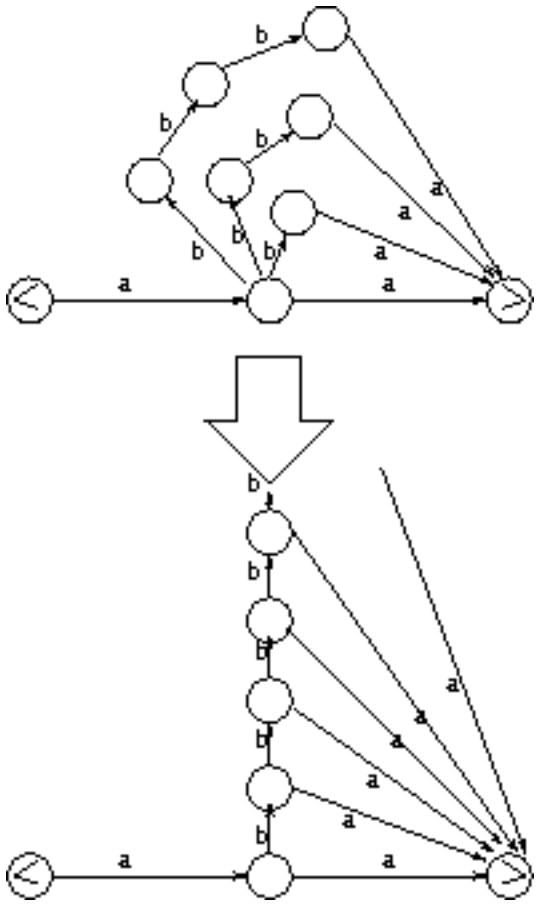
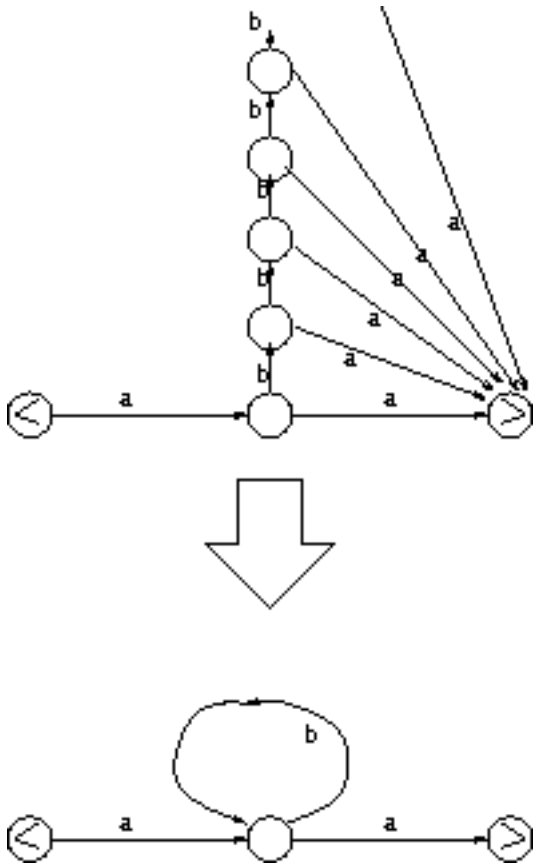


Figure 7.5: The final gluing.



Finally, in *fig*.7.5 we amalgamate the tower into a self loop, and all the final arcs also get amalgamated. This yields the necessary finite state diagram for the language.

All that remains is to carefully formulate the gluing rules which make the transition from the original diagram to the final diagram. It seems reasonable that for finite state languages there *are* such rules; what we need is a procedure which tests to see if the language generated by the transformed network is altered by the gluing operation. This looks to be a finitary business.

The details may be found in Eilenberg's excellent *Automata, Languages and Machines*, Vol.A. Chapter III section 5, although the terminology requires some translation. Note the reference to Beckman's IBM Research Report.

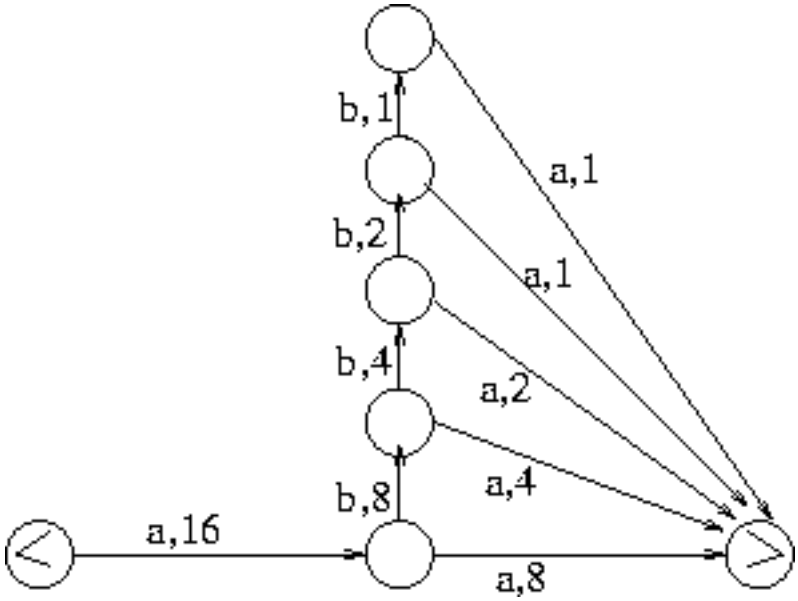
The ideas outlined can be turned into operations on ReWrite grammars, where the methods rapidly become virtually unintelligible. It is much easier to visualise them in terms of finite state diagrams, or *automata* as Eilenberg calls them. In the ReWrite grammar form, we have just outlined the ideas for inference of finite state grammars which King Sun Fu described in the papers cited in the bibliography at the end of this chapter.

For the stochastic case, there are some interesting issues: you will recall our inference of a local grammar for the stochastic language $\{(ab^n a, \frac{1}{2^n}) n \in \mathbb{Z}^+\}$. The same process can be applied to a finite

sample. It can also be applied to the right segments of a stochastic language obtained by doing identifications from the left, as above. All that is necessary is to keep counts of the number of strings and to add them when we do the amalgamation. Then if we had the sample:

$\{(aa,8),(aba,4),(abba,2),(abbba,1),(abbbba,1)\}$ where the numbers give the string occurrences, we may do the same process of identifications, to give the diagram of *fig*.7.6 where the counts on the arcs have been obtained by summing.

Figure 7.6: The Stochastic case.



Now the diagram has merely summarised the data given, and does not produce anything essentially new. Nevertheless, it is tempting to collapse the column, even though it has finite height. We might justify this by observing that from the top of it, the view backward looks, if we reduce the counts to fractions, exactly the same as the view from the topmost but one-provided we do not look too far back. If we look at most two nodes back, the top of the tower sees behind it a 0.5 chance of coming in from its predecessor by emitting a *b*, and an equal chance of going home by emitting an *a*. The predecessor sees exactly the same, as does its predecessor. This suggests that amalgamating the top of the tower with the node preceding it will be harmless, and we might just as well identify the arcs leaving, because they go to the same place with the same symbol. This gives a self loop at the top of the tower. Of course, it is only locally harmless. It actually changes the language generated; it increases it to an infinite set. And having done this, we now discover that the view forward from every node of what is left of the tower is the same as for every other such node, so we might as well amalgamate them all.

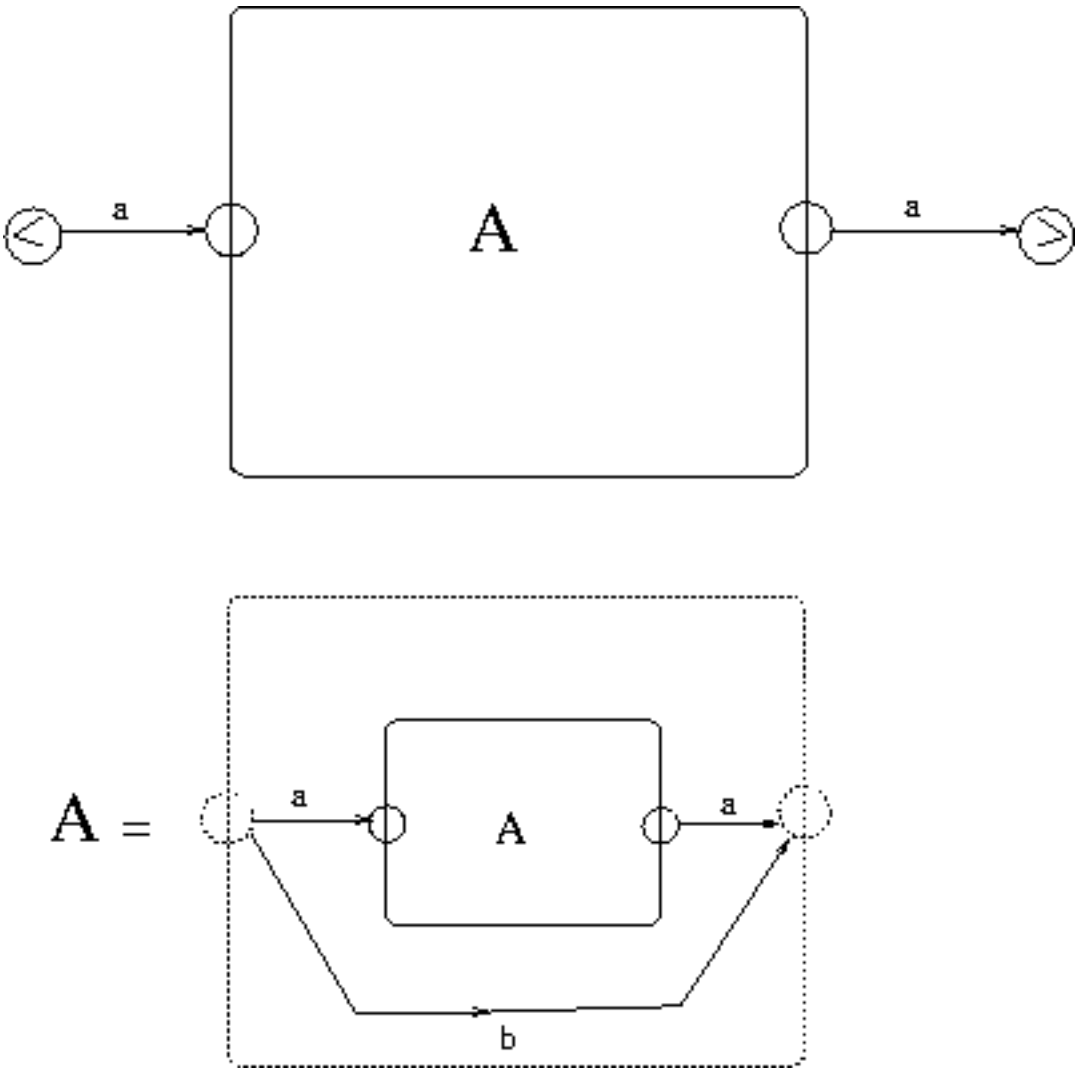
We see then that we can compress the diagram just a little more than is altogether good for it, and if we do, we get a model out which expects other things than those we actually observed. We could do even more compression and get out even larger languages: in the limiting case we blithely identify every node, and we expect any non empty string on the alphabet $\{a,b\}$. If there are *p a*s and *q b*s, the probability is $\left(\frac{32}{47}\right)^p\left(\frac{15}{47}\right)^q$. The reader is encouraged to satisfy himself that the result is a well defined stochastic

language and that all the probabilities sum, in the limit, to 1. This might suggest a neat way to work out some of those formulae involving multinomial expansions which are so tiresome to remember.

This can be done with any language sample on any alphabet, the business of amalgamating nodes and summing counts can be performed with more or less enthusiasm and corresponding disregard for the actual data. This gives an alternative to seeking hidden markov model intialisations by going into seedy bars if you prefer, for some reason, to avoid them. It is quite entertaining to experiment with some wild language samples (such as you might get out of a Vector Quantiser for speech) and see if you can find interesting and plausible rules for amalgamating nodes.

It is possible, as was indicated earlier, to modify the finite state diagrams or automata for the case of Context free grammars; the idea here is to allow boxes to contain copies of themselves as well as being part of a network. The case of the language $\mathcal{L}_{cf}$ discussed above is shown in such a form in *fig*.7.7.

Figure 7.7: A Push-Down Stack Machine.



Inference for such things is possible in special cases, but one might reasonably doubt if this is a problem that is likely to occur in practice. If we think of the problems faced by a cockroach trying in his limited way to remember a sequence of muscle contractions, say, and if we suppose that allowable sequences that have successfully got the 'roach where he wanted in the past have had some particular form, then we observe that although some small amount of palindromic structure is conceivable, it seems, *a priori*, somewhat unlikely that it would get carried very far. And we are most likely to be concerned not with the case of a deterministic grammar or language, but a stochastic one, and with a finite sample. The reader might like to try the following experiment: construct two grammars for a small alphabet, generate a dozen strings from each, put the strings from one grammar into a red box and from the other grammar into a blue box. Procure a small child of age three or thereabouts. Read the strings from the red box grammar while ostentatiously pointing to the red box, then read the strings of the other grammar while pointing to the blue box from which they were drawn. Now generate a few more strings, and ask the child to point to the appropriate box. See if the child gets the right answer. If not, is it colour blind? Did you make an injudicious choice of grammars? Try some other strings not generated by either grammar and see if the child is uncertain. The child is doing grammatical inference on the sample provided, and it is of some interest to speculate about the class of grammars it is prone to use. If instead of reading out symbols and making noises, a sequence of brightly coloured blocks is used, does the child still use the same grammatical inference process?

Return child after use.

The grammatical inference problem can be recast to other forms, and other problems, notably a large fraction of those occurring in I.Q. tests, can be viewed as variants on grammatical inference. Short sequences of distinguishable events often occur in nature, and classifying them is something which may be rather desirable for an organism wishing to reproduce itself before being eaten or starving to death. It would be rather nice to know to what extent the central nervous system of animals uses consistent methods for obtaining such classifications, or equivalently, whether some particular class of stochastic grammars is preferred. I shall return to these somewhat speculative matters later when discussing a new class of neural models.

Next Up Previous Contents

Next: Streams, predictors and smoothers **Up:** Alphabets, Languages and Grammars **Previous:** Grammatical Inference *Mike Alder*
9/19/1997

Streams, predictors and smoothers

A language as we have defined it is a set (usually infinite in principle) of strings. It may be noted that natural language fits (uneasily) into this definition in several different ways; it may be taken with alphabet the printed characters, in which case we can take the words as the strings, or the sentences, or even the paragraphs. If we take the words, then the strings of one language become the symbols of the next, where the strings may be sentences. Or paragraphs, or chapters, or books, or for that matter libraries... So natural languages are actually hierarchies of languages as we have defined them, which suggests that we may not be defining them very intelligently.

What we are confronted with on opening a book is, after we have mastered the convention of scanning from left to right across a line, down a page and up to the next page or turning over, a stream of characters, or a very long string indeed. Since we don't read it all at once, and don't know how long it is precisely, it makes sense to idealise it as we do a signal received from a television transmitter, as an infinitely long sequence of things. In this case as an infinitely long sequence of symbols. This is how text files are thought of as stored on disk in a computer, although they are not in fact, happily, infinitely long. Then a white space is a *separator* symbol and a full stop, `.', is another, while linefeed carriage return, twice, is another. This allows us to chop the stream up into strings which then provides us with a language, in fact several neatly nested.

Definition

A *stream* or *time series of symbols* on an alphabet A is a map from $\mathbb{Z}^+$ into A.

A local grammar of order *k* makes just as much sense for a stream as it does for a stochastic language. In fact if we are given a sequence of strings from a language, we can turn them into a stream by either putting begin < and end > symbols around them and just concatenating, or perhaps replacing >< by a white space in this arrangement to save on symbols.

Definition

A *stream sample* on an alphabet A is a map from the set [1..N] to A, where [1..N] is the set of all positive integers less than N+1.

In real life, we are given stream samples. N may be several million or more. A may be $\{0,1\}$ in computer applications. A local grammar becomes a *predictor* of what the next symbol may be as it moves along (up? down?) the stream.

Definition

A *predictor* of order *k* for a stream over A, is a map from the set of *k*grams in a local grammar for the stream to the set of pdfs over the alphabet A.

In other words, if I give you *k* symbols which have been found to occur consecutively, a predictor of order *k* will accept this as input and return a probability, conditional on the input, for such a *k*gram being followed by any given symbol of A.

Example

Go through the stream or stream sample and obtain a local grammar of order 2 by listing all trigrams. Now collect together all trigrams which agree in the first two places, and count for each symbol in the alphabet the number of times it appears in the last place. For example, the preceding two sentences would have the trigrams

```
_an
_al
_ag
_a_
```

which start with a whitespace (replaced here by an underscore) and have `a' as the second symbol. The first and second occur twice and three times respectively. Thus the probability estimate of getting ` an' is $\frac{2}{7}$.

Definition

The predictor which stores *k*+1grams by storing all those with the first *k* symbols the same and retaining a count of the different possibilities for the last symbol is called a *local predictor of order k*.

It is easy to see how such a predictor could have its uses in an OCR program. If you were to collect trigrams from the text you were sure about, you could compare with known statistics for English and decide if it was English, or perhaps some related language such as bureaucrats gobbledegook. Then once you had a plausible identification of the language, you could clean up `Profes5or' by using two sets of data, the relative probabilities of `es5', `ess', `esa', and so on from the probabilistic model for the character, and the relative probabilities from the predictor in the absence of any actual information about the character except its context. A little thought will show how to express the net probability of the character being a `5' or an `s'. This is smarter than using a dictionary, since it can cope with new words (such as proper names) which do not occur in the dictionary. Such a system will easily improve the performance of an OCR system. Indeed it can be used to build a font independent OCR device as follows:

Suppose that the text can be segmented into characters with high reliability and that the object language is known. I scan the text and start with the first character. I assign it at random to one of the letters of the alphabet. I move on to the next character. If it is the same as one I have seen before in this data, I assign it to the same letter as last time, if it has not been seen before, I assign it to a new letter so far unused, at random. I continue in this way until all characters have been assigned letters. This is simple clustering.

I have now got the text transcribed to ASCII characters, but the text has been coded by a `Caesar' code, that is a 1-1 substitution scheme. Now I use the known statistics of the language to break the code, a

matter of a few minutes work by hand, and much faster on a PC.

Given a book of English text to scan, with the text stored as letters not characters, the little green *thing* from outer space would be struck at once by the fact that one character is much more common than any other. I am referring not to the letter `e', but to the white space separating words. These characters, used by us as separators, may be identified as separators without prior knowledge. This is just as well; in speech there are no acoustic cues telling the auditor when a word has ended- there are no white spaces. It would be easy enough to put them back in were they left out of printed text however. How this may be done will be described next.

Next Up Previous Contents

Next: Chunking by Entropy **Up:** Discrete Dynamic Patterns **Previous:** Inference of ReWrite grammars

Mike Alder
9/19/1997

Next Up Previous Contents

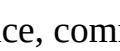
Next: Stochastic Equivalence **Up:** Discrete Dynamic Patterns **Previous:** Streams, predictors and smoothers

Chunking by Entropy

Suppose we run a local predictor of order *k* over a stream sample and consider at each point in the stream the probability distribution or more generally the *pdf* over the alphabet at each stage. One of the important things we can say about the predictor is what it's entropy is at each point of the stream. Another thing we can do is to compute the amount of `surprise' which the actual symbol produces at each point of the stream. We can also compute the mean entropy of the stream sample relative to the predictor by summing the last number over the entire stream sample and dividing by the number of symbols, and we can also compute the mean entropy of the predictor. If the predictor is a good model for the stream, these will be about the same. If one predictor is better than another, we can tell by comparing the mean information (surprise) supplied by the stream to each predictor, and picking the smaller number. God, who is omniscient, is never surprised by what happens, which must make for unspeakable boredom.

It was observed by Shannon that as we run along English text, the entropy of a loal predictor of order *k* for $k > 1$ decreases inside a word until it comes to the white space. If I give you as a (*k*-1)gram,

the number of possible next symbols is large and the probability distribution is broad. If you get

as the (*k*-1)gram, the number of next symbols is somewhat less and their probabilities somewhat higher, so the entropy is less. At there are very few next possible symbols in English, and *m*, *p*, are much more likely than the others. At *comp* the next symbol is a vowel and the entropy is very low. At *ring* the next symbol is either white space, comma, *s* or *e* to high probability. And at  it is virtually

impossible to say what the next symbol will be and the entropy has shot up again. So even using 5grams to obtain a local predictor, it is easy to find separators without prior knowledge of which symbols have been selected for that purpose. Alternatively, if someone were to go through text and remove all the white spaces, it would be possible to go through and do a reasonably good job of putting them back in again. Or to put it in a third form, we can use a predictor to go through text and chunk it into words.

Definition

Using a predictor to separate a stream of symbols into consecutive strings as described above is called *entropic chunking*, and the strings so found are known as *chunks*.

Once chunks have been found by entropic chunking, the stream of symbols may be reconstructed to become a stream of chunks.

Definition

The process of mapping a stream of symbols into a stream of chunks is called *UpWriting*; we say that the

stream of chunks is at a *higher level* than the original stream of symbols. Recovering the stream of symbols from the stream of chunks is called *DownWriting*. UpWriting entails compiling a dictionary of chunks and assigning a new symbol to each chunk. DownWriting entails looking up the dictionary and replacing each chunk with the corresponding string of symbols.

It must be conceded that the practicalities may be considerable; finding maxima in the entropy may give not words of English but stems and inflections. Thus the *s* at the end of a word used to denote plurality may be a separate chunk, and similarly with *ed* to denote tense. Inflected languages simply haven't evolved to the consistency of having a white space used to denote a chunk end.

Experiments make it convincing that the method of entropic chunking by a local predictor works sufficiently well to be applied to machine language programs, *inter alia,* and that they discover the obvious structure of operators and addresses.

As well as chunking by entropy, we can chunk by the amount of information supplied by a character, or the *surprise* as I have called it earlier. This will, by definition, give the same average result as chunking by entropy if our predictor has been obtained by counting occurrences in the text itself, but it can give different results on different texts.

The process of UpWriting yields compression; it is easy to see that knowing the language in which something is written gives a large amount of redundancy to be extracted and that this dictionary method will accomplish something of that extraction. It is also easy to see that there may be a language, or more exactly a stream, for which chunking doesn't work. Let it be noted that there do exist streams for which it does work, and that natural language texts are among them.

The process of UpWriting a stream into a higher level stream may be iterated. In the case of natural language streams, this yields a progressive decomposition into larger and larger chunks, as has already been described. In each stage there is some compression of the original stream, and each stage can be implemented in the same way by a local predictor of fairly low order. It is easy to persuade oneself that one attains greater compression than by having a high order predictor at least for some kinds of stream. Let us call such streams *hierarchical* streams.

Formally:

Definition

A stream is said to be *hierarchical* if it allows compression by a sequence of UpWrites by low order predictors.

Since it is apparent that much human information processing involves a hierarchy of structures, hierarchical streams are natural things to study in that they are relatively simple cases of the decomposition process that, presumably, extends to learning to drive a car. The existence of clichés tells us that English is at least partly hierarchical to two levels, but plainly there is more going on than that.

Next Up Previous Contents

Next: Stochastic Equivalence **Up:** Discrete Dynamic Patterns **Previous:** Streams, predictors and smoothers *Mike Alder*

9/19/1997

Stochastic Equivalence

Consider the natural language English with symbols at the word level. Take a stream, any stream, and consider such sentences as `*The cat sat on the mat*'. Regarding this as a *k*gram for *k*=6 suppose we collect all other such 6grams which agree with the given one except in the second symbol. Thus `*The dog sat on the mat*' will be in the list. We have something very like the local predictor of order 6 except that we take account of context on both sides. We will get a similar object out, a probability distribution over the alphabet (in this case dictionary) of symbols. Of course, I didn't have to pick the second place as the `variable', it could have been any location.

Definition 14505

A *context* of order *k* for a stream $\mathcal{S}$ over an alphabet A is a string of the stream together with a designated place between 1 and *k* which is replaced with a placeholder symbol. The symbols of the alphabet together with their relative frequencies which match the place holder when all the other symbols of the context are matched define the *context distribution* for the stream and the context.

Suppose I have two context distributions arising from different contexts. Then it might happen that they give rise to distributions which are pretty much the same; as distributions, they may not differ by more than I would expect from simply sampling variation. There are various measures of the difference between distributions, from χ^2 to Kullback-Leibler via a straight euclidean metric, and I shall not go into any of them here. It suffices to observe that it makes sense to decide that some context distributions are not distinguishable from others. It is clear that this will happen in English; a cat and a dog are both things that can sit on mats, but they can do quite a lot of other things in common too. So the context distribution for `The -?- sat on the mat' is probably not discriminable from that for `She played with the young -?-' for most stream samples. If we have a distribution arising from a context or a local predictor (a particular sort of context with the place holder at the right hand end) which is not well represented because the context is a rare one, this could be useful as we could get a better estimate of the true distribution by pooling with a distribution obtained from another context. This is fraught with risk, of course, but there is a risk involved in having too few data to make a reliable estimate. Life is pretty fraught all round.

At the letter level in English text, there are some very distinctive distributions which may be found by examining different contexts: separators and punctuation symbols for example, and digits also form a cluster in the space of probability distributions over the alphabet of symbols. (Which looks exactly like a piece of $\mathbb{R}^n$, where *n* is the size of the alphabet). Vowels form a distinguishable cluster in this space, with a consistent ratio of counts in a variety of different contexts. And some reflection suggests that this feature of natural language is not confined to English, nor is it confined to the letter or word level.

Any such regularity as this can be exploited in a grammar (regarded as a data compression device for specifying streams) or language model. Suppose I were to assign labels to the distributions. If there were too many of them and no grounds for pooling them, then I could have as many distributions as contexts, which is many more symbols than my alphabet originally contained. If, on the other hand there are very few, it might be desirable to do a new sort of UpWrite to the names of the distributions. The DownWrite from this would require making some choice of which symbol from the distribution to select. Instead of storing the symbol names and assigning them equal length, we would do well to send the distribution and a derived coding for the symbols exploiting the fact that commonly occurring ones should have short names. This could be a useful feature of any device which had to remember lots of symbol strings. A stream or language might have the property that it would be more economical to code clusters and places within the clusters than just to send the stream, it would depend on the extent to which clustering in the space of context distributions occurred.

Example

The stream is obtained using 16 symbols *a* through to *p*. There are auxilliary symbols A,B, C,D which are `bags'. A is the bag containing *a*, *b*, *c* and *d*, and so on down to D which contains *m*, *n*, *o*, *p*. The bags are selected cyclically, A,B,C,D,A,B,C,D.. until 1024 have been listed altogether. Each time a bag is selected, one of the symbols inside the bag is picked with equal probability one quarter. So something like *aeimbfjnc*.. comes outof the process. Now it takes 4 bits to specify one of the 16 symbols, and there are 1024 of them, so a straight listing of the stream sample would occupy 4*K*bits. But this does not exploit the fact that the bag structure is cyclic. If we have four bags, A,B,C,D then we can code each symbol by coding the bag it comes from with 2 bits, and the symbol in the bag by another 2 bits. Now the cyclic bag structure allows us to specify the bag 2-bits for each bag in much less than 2*K*bits: we need to code the information that the cycle ABCD recurs 256 times. A thousand bits or so would seem more than adequate for this. So we save at least 1*K*bits by using the structure of the stream.

The discerning and reflective reader will note that this may be thought of as a *Hidden Markov Model* for the language. The `bags' are the hidden states, and the state diagram is particularly simple.

Definition

A stream is said to have a *stochastic equivalence* structure when it is the case that the context distributions for some order *k* of context form clusters in the space of probability distributions over the alphabet. If two symbols *a* and *b* occur with similar probabilities in a significant number of distributions which are all close in the space of all distributions over the alphabet, we may say that *a* and *b* are *stochastically equivalent*.

It is clear, just by stopping and thinking for a while, that English at the letter level has clusters for digits, for punctuation and for vowels. It is easy to confirm, by computation on English text, that for $k = 4$ these clusters actually exist. It is rather more expensive to compute the distributions for English at the word level, particularly for large *k*, but it is possible for $k = 3$, and again one's intuitions are borne out: synonyms and also antonyms are discovered sitting in the same distribution; class name exemplars seem to be implemented as elements of such distributions. For example, the class of domestic animals (including children) constitutes a distribution which occurs in many texts in many different contexts. Buses, taxis, cars, boats and aeroplanes figure as prominent elements of a distribution which again recurs in many stream samples and many contexts. This may not be altogether surprising, but it is hardly a necessary feature of a stream.

The above definition lacks the precision which one might reasonably expect in book written by a mathematician, even one who has gone downhill into engineering, and it is a matter of some irritation that it is not easy to give a satisfactory definition of what it means for a set of points to be clustered. (A probability distribution over an alphabet of N symbols *is*, of course, a point in $\mathbb{R}^N$.) There are many different definitions, see any of the books on clustering, and althought they usually agree on obvious cases, they may difer on the marginal ones. For present purposes, I suggest the reader thinks of a modelling by a mixture of gaussians and uses Rissanen's stochastic complexity criterion to decide on the optimal number of gaussians. If the optimal number is zero or one, then we can say that there is no clustering. If the number is small compared with the number of data points so that many distributions find themselves packed closely to other distributions, then we obtain a working definition of stochastic equivalence. I do not propose this as any more than an *ad hoc* heuristic at this point.

If there is a clustering structure for the context distributions, then we can take all the *k*gram predictors which have `close' distributions in the last place, and deem them to be `close' also. After all, they ought to be stored together in any sensible prediction scheme precisely because they predict the same sequent. So we can cluster the *k*-1grams. This gives us the possibility of a new type of UpWrite, a stochastic equivalence class of *k*grams gets written to a new symbol.

Definition

An UpWrite may be of two types: the *chunking UpWrite* assigns a symbol in the UpWrite stream to a *substring* in the base stream. The *clustering UpWrite* assigns a symbol in the UpWrite stream to a *stochastic equivalence class* in the base level stream.

Next Up Previous Contents

Next: Quasi-Linguistic Streams **Up:** Discrete Dynamic Patterns **Previous:** Chunking by Entropy *Mike Alder*
9/19/1997

Quasi-Linguistic Streams

Definition

A stream is said to be *quasi-linguistic* if it is hierarchical and at least some of the (chunked) UpWritten streams admit a stochastic equivalence structure. In other words, we allow ourselves to do both kinds of UpWrite, into chunks of consecutive symbols or into stochastic equivalence classes of symbols. If any sequence of either UpWrite yields data compression, then the stream is *quasi-linguistic*

Although the definitions are not as tight as is desirable, they are adequate for there to be reasonable grounds for consensus on whether quasi-linguistic streams exist and whether a particular stream is an example of one. Now it is easy to convince oneself by a certain amount of simple programming that samples of English text of less than a million characters give reasonable grounds for believing such things exist. There are many other possible structures which may be found in Natural Language stream samples, but whereas grammarians have been usually concerned with finding them by eye and articulating them in the form of rules, I am concerned with two other issues: first the question of inference, of how one extracts the structure from the sample and what algorithms make this feasible. Second the investigation of more powerful methods of specifying structure than by rule systems. It should be plain that it is certainly a structural feature of natural language if it exhibits a stochastic equivalence structure, but it is not one which is naturally articulated by giving a set of rules.

If such a structure is found in a stream, then it can be used for smoothing probability estimates of predictors.

Suppose for example we are using a trigrammar at the word level to constrain the acoustic part of a word recognition system, as is done at IBM's Thomas J Watson Research Center in Yorktown Heights. It is common to encounter a new word never seen before (proper names, of ships, for example) and also common to encounter two words never seen before in that order. Suppose we have a string *ABCDEF*? and want to predict the ? from the preceding words, *EF*, but have never seen *EF* together, although we have seen *E* and *F*. We may first try replacing *E* by something stochastically equivalent to it in the context *CDE*. There are, presumably, a number of other symbols which might have occurred in place of *E* after *CD*. We identify the probability distribution as belonging to a cluster of other such distributions. We replace the given distribution by the union of all those in the local cluster. This gives us a list of other symbols which are stochastically equivalent to *E*. We weight them by their relative likelihoods, and list them in the form $CDX_1, CDX_2, \cdots$.Now we take the *F* and replace it by all symbols Y_i which are stochastically equivalent in the context DX_j. Again we weight by their probabilities, and include the DX_jF case as of particularly high weight. We take the union of the resulting distributions for the weighted X_jY_i as the bigram predictor. Then we try to recognise this distribution, i.e. we choose the closest cluster of known distributions. This is our final distribution estimate for the successor of

ABCDEF. Similar methods apply whenever *E* or *F* is a new word which has never been seen before.

If one reflects on the way in which we use context to determine meaning of a new word in a page of text, it may be seen that this method holds out some promise. Semantic meaning *is* inferred from other known meanings and the syntax of language. If we take the narrowest of stochastic equivalence sets, we may replace some of the symbols in a context with a stochastic equivalence class, and generate a larger stochastic equivalence class by the process described above. These classes become very large quite quickly, and rapidly tend towards things like the grammatical categories `Noun', `Adjective', `Transitive Verb', and so on.

Quasi-linguistic stream samples are of interest largely because they constitute abstractions of natural languages, and moreover the abstractions have been constructed so as to facilitate the inference of structures. This `Newtonian' approach to language analysis differs from the traditional `scholastic' approach, which you may find described from the writings of Chomsky on. The question arises, to what extent do streams of symbols occur, other than in natural language, which exhibit this structure?

A real number is a symbol, and so a time series of real numbers may be considered in this manner. There is of course a metric structure on the symbols which is usually presumed to be known. The interested reader is invited to investigate extensions of syntactic methods to real valued time series.

Next Up Previous Contents

Next: Graphs and Diagram Grammars **Up:** Discrete Dynamic Patterns **Previous:** Stochastic Equivalence

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Exercises **Up:** Discrete Dynamic Patterns **Previous:** Quasi-Linguistic Streams

Graphs and Diagram Grammars

King Sun Fu, who was an engineer working in Pattern Recognition and not an oriental potentate, was one of the first to note that significant pattern recognition is syntactic and his works may be consulted for examples, some rather artificial, of cases where syntactic methods can be applied. Perhaps the classic case is an attempt to segment handwritten characters into stroke primitives, and then to extract the syntax of the sequence of primitives. The primitives were extracted by eye, and may not have been the best choice.

The difficulties engendered by the segmentation problem have stimulated attempts to go to higher dimensions than one and to apply to more complicated entities than symbol strings, the same kind of syntactic ideas. A *directed graph* is a set of nodes with arcs going between some of them, an arrow showing the direction of the arc. A graph is a directed graph with arcs always going both ways, or alternatively with the arrows left off altogether. It is easy to see that you can glue some such objects together to get others, and that there are thus some generalisations of concatenation for strings. It is plain that some sort of segmentation and decomposition of graphs (directed or not) can be done and that rules for allowing some subgraphs to be rewritten to others, just as in the standard rewrite grammars, are possible. What is not apparent is whether the things arise in nature with such decompositions built in. It is not plain how you make a predictor or a context, and without these you cannot easily get chunking or stochastic equivalences. The opportunities for Pure Mathematicians to occupy their time with little profit to the rest of us would seem to be excellent, but people have thought that before and been horribly wrong.

Huffman, Clowes and later Waltz have analysed the way in which trihedral edges in line drawings can be constrained when the drawings depict solid objects such as cuboids. This is a nice example of syntax arising naturally in the world, but the simplifications needed to make the subject tractable reduce the inference problem to triviality.

McLaughlin and Alder have shown how some structure in a silhouette of a hand, namely that it is built up out of fingers and a palm, can be extracted automatically and used to distinguish between left and right hand palm prints. The method has also been used to classify aeroplane silhouettes and to learn the rotation, scale and translation transformations. This will be discussed at greater length in the next chapter.

Next Up Previous Contents

Next: Exercises **Up:** Discrete Dynamic Patterns **Previous:** Quasi-Linguistic Streams *Mike Alder 9/19/1997*

Exercises

1.

Get one of the children's reading primers giving details of the adventures of Janet and John or Dick and Dora, or Peter and Jane or some similar pair. Put the sentences into the computer.

Alternatively find such a source on the net if you can, and then tell me what it is. Note that teaching these sentences to a computer program is not at all the same as teaching them to a child because (a) the child has already acquired a spoken language and (b) the child can look at the pictures that come with the text and interpret them. The program can't.

Nevertheless, constructing a program which can learn some of the structure is worth a try.

First write a program which takes the sentences as strings at the letter level. Construct a *trigrammar*, a list of consecutive triples of letters, including whitespace and punctuation. Use it to build a prediction model for the data, and determine how to chunk letters into words. And other things. Cope with the problem of new bigrams by the `fallback' method, ignore the first letter. If the last letter of the bigram has never been seen before, fallback all the way to letter frequency. Compute the entropy of the trigrammar model of the text. Compare with the mean information supplied to the model by a piece of text it hasn't seen before.

Repeat for the word level. It may be convenient to have punctuation regarded as words. A vocabulary of not more than about 100 words is advisable. Again, compute the entropy of the model of the data. If you obtain estimates by trying it out on new data, It will be necessary in both cases to cope with the situation where a new symbol is encountered: it is suggested that you reserve a $\frac{1}{n}$ probability for anything you haven't seen before, given that you've seen n different things so far.

Classify the words into types by eye. Names of children, for example, form a subcategory. UpWrite the original text at the word level into `bags', each bag containing the terms of a category such as **childname**. Now model the UpWritten stream. Determine the entropy of the model of the UpWritten stream. Determine the entropy of the model for the original stream consisting of the bag stream and the DownWrite.

2.

Make up a bag grammar by choosing some particular sentence structure such as

Article < Adjective < Noun < Verb < Article < Noun < .

Now generate a list of possible words in each category, not more than twenty words in each. Finally generate sentences from any choice of a *pdf* for each `bag'.

The problem is to try to recover the bag structure from the sample. Try it as follows: Take a trigrammar and for any words *ab* list the sequents of *ab* and make them into a probability distribution over the alphabet. Now in the space of all such distributions, look for clusters. If you find at least one cluster, label it *C* and to assign it the group of *pdf*s in the cluster. Now perform a *cluster UpWrite* on the original stream by replacing whatever follows *ab* by *C* precisely when *C* is the cluster containing a distribution obtained from collecting sequents to *ab*.

Replace the old by the new stream, and do it again. How much data is required before the categories of `noun', `verb', emerge? What is the entropy of the model of the stream which first predicts bags then predicts elements in a bag by assuming the probabilities for choice of bag element are independent and the mean *pdf* for the cluster? How does this compare with a trigram word model?

3.

Automate the process of picking bags such as **childname** or **animalname** in the case of data taken from the children's primers. This could be described as `real data', almost. Do you get any compression using these methods? Do you get better results than an *n*grammar for various *n*?

Next Up Previous Contents

Next: Bibliography **Up:** Discrete Dynamic Patterns **Previous:** Graphs and Diagram Grammars *Mike Alder*
9/19/1997

Bibliography

1.

S. Eilenberg, *Automata, Languages and Machines*, Vols A,B. Academic Press 1974.

2.

S. Ginsberg, *Algebraic and automata-theoretic properties of formal languages* American Elsevier Pub. Co., 1975.

3.

J. Hopcroft and J. Ullman *Introduction to Automata Theory, Languages and Computation* Addison Wesley, 1979.

4.

H. Ehrig, H.-J. Kreowski, G. Rozenberg (eds.) *Graph Grammars and Their Application to Computer Science* : 4th International Workshop, Bremen, Germany, March 5-9, 1990.

5.

P.S.P. Wang (Ed.) *Array grammars, patterns and recognizers* World Scientific, 1989.

6.

Frederick J. Damerau, *Markov models and linguistic theory : an experimental study of a model for English* Mouton, 1971.

7.

G. Herdan, *The advanced theory of language as choice and chance* Springer-Verlag, 1966.

8.

The *IBM Language Modelling Kit* is a set of papers produced by the Continuous Speech Recognition Group, IBM Thomas J Watson Research Center, Yorktown Heights, NY 10598, U.S.A. Some have been cited earlier.

Mike Alder
9/19/1997

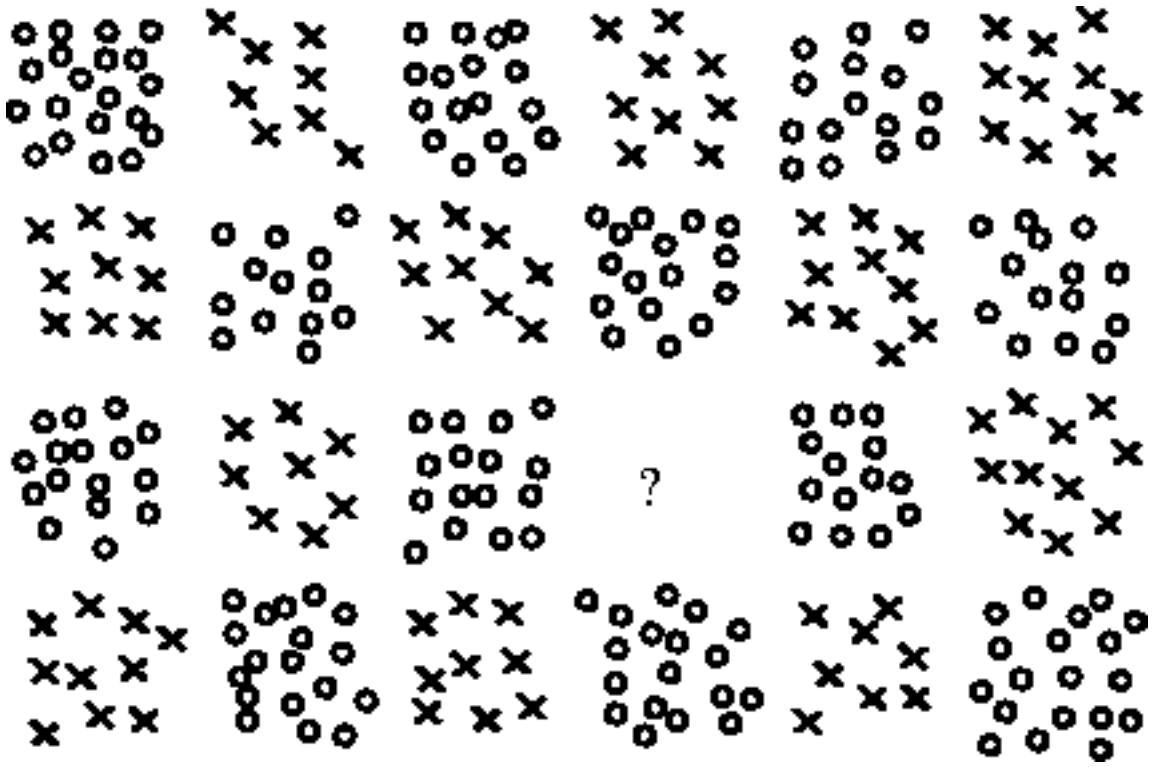
Syntactic Pattern Recognition

Returning now to the principal thread of static Pattern Recognition, and armed with the righteousness that comes from having read up a little algebra and the attendant feeling of virtue, we contemplate a few uncomfortable facts.

The first is the practical difficulty attached to segmenting before measuring and classifying. It is common to explain to outsiders something of the difficulties of pattern classification and have them explain to you that standard function fitting methods will work. When you draw a `5' and do so with the horizontal stroke not connected, and ask how they propose to deal with this case, they shrug in irritation and assure you that these little details may be taken care of by a variety of means, none requiring intelligence. They hint that simple bottom-up procedures, involving minor extensions in the direction of existing line segments, will take care of the difficulties. They are wrong. Their eyes tell them it is perfectly in order to make a join in some cases but not in others, but close inspection reveals that they have already made the classification by this point. It is not done bottom-up at all; or if it is, there are elements, are smaller than characters but bigger than pixels, which are the basis of recognising a `5'.

Segmentation is not, in short, something to be done independent of the classification process. But this seems to provide us with a chicken and egg situation. How do we decide when things belong together in order to feed them into a classifier, without classifying them? This is a very profitable question to think about, and it is left to the reader to do so.

Figure 8.1: A classification problem every system discussed in this book, so far, will get wrong.

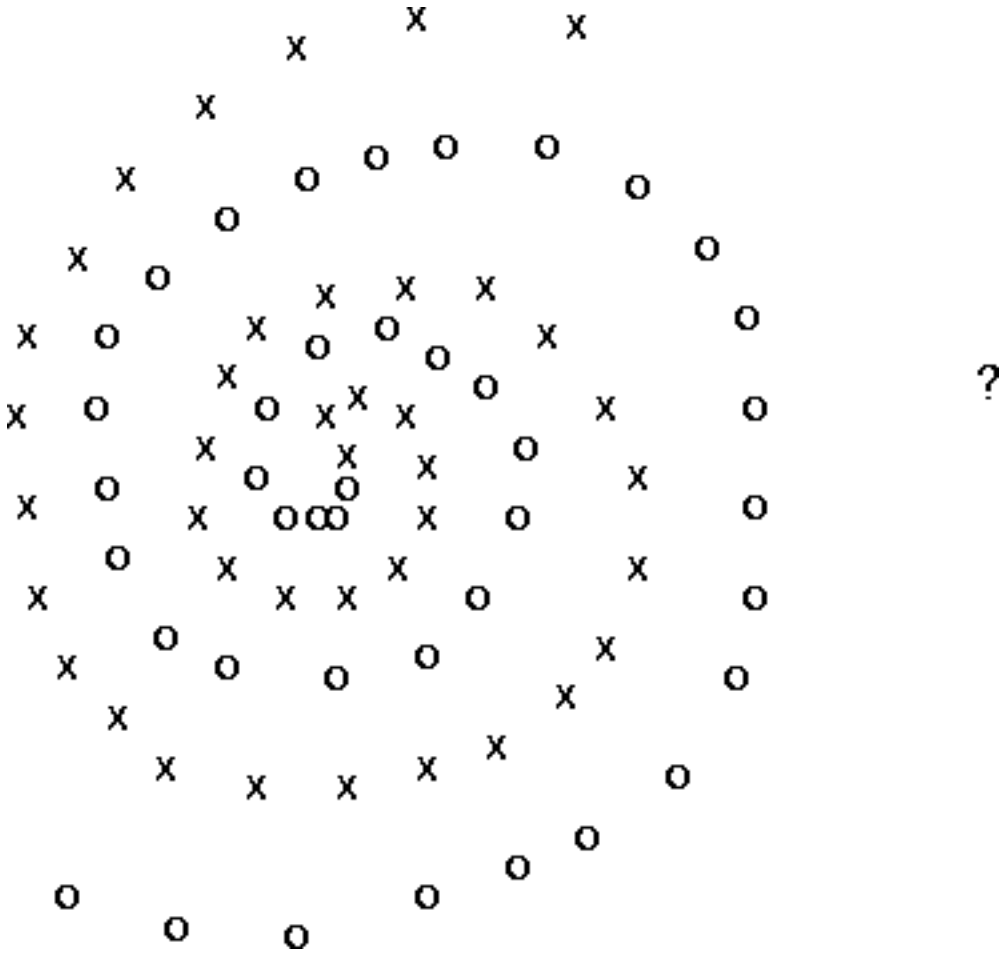


The second is more prosaic and less abstract. Contemplate the simple diagram, of two data sets in the plane, shown in *figure* 8.1. I have labelled the two classes as noughts and crosses out of a fondness for the game known to the English Speaking World as Noughts and Crosses, and to the American Speaking World as Tic-Tac-Toe.

Now focus on the point at the dot belonging to the question mark. Would you expect this to be a nought (O) or a cross (X)? Note that although there is no answer to this question, since I am the one who generated the data and I am capable of anything, most people would agree that the category they would be most inclined to expect is a cross, despite the fact that every method of pattern classification we have discussed would give the opposite answer. The same holds for *fig*.8.2, where most right thinking people would expect the ? to be a cross, despite the fact that its closest neighbours are mostly noughts.

The fact that the automatic systems discussed so far would all give different results from the eye of a child, merits a little thought. It suggests something essentially circumscribed about our methodology.

Figure 8.2: Another one.



It is, of course, easy enough to devise a modification to the existing methods which will give the `right' answer. This is to miss the point. We aren't looking for a quick fix; we thought we had the business of pattern recognition reduced to finding clusters and suddenly we discover that life is much harder than that. So we need a fix that will work not just on the two figures shown, but on lots more which may be sprung on us any minute now. We need to ask ourselves what it is about the data sets that means our simple systems don't work.

Contemplate a very similar situation: The word `profes5or', where the `5' is replaced by an `s', despite the low level data suggesting that `5' is what is there. In all cases, there is a low level or neighbourhood expectation, and a higher level of structure to the data which is given precedence by the human eye. In the case of `profes5or', the level above the letter level is the word level. In the case of *fig*.8.1 the level above the local is the alternating or chess-board pattern to the data. In the case of *fig*.8.2 it is the spiral curves which are extrapolated by the eye. I claim that in all cases, it makes sense to say that there are two levels of structure and that the higher level is accorded precedence in the human decision as to which class should be assigned to a new point. I claim that just as there is a process of UpWriting strings of letters into words, there must be a way of UpWriting sets of points in the plane into something else, so that we have some basis for making decisions at a different level. I claim that the intuitive idea of levels of structure, which can be shown to make good algorithmic sense for natural language can also be found to make sense in the case of more general, geometric data.

- Precursors
- Linear Images
- Curved Elements
- Parameter Regimes
- Invariance:
  Classifying Transformations
- Intrinsic and Extrisic Chunking (Binding)
- Backtrack
- Occlusion and other metric matters
- Neural Modelling
  - Self-Tuning Neurons
  - Geometry and Dynamics
  - Extensions to Higher Order Statistics
  - Layering
- Summary of Chapter
- Exercises
- Bibliography

Next Up Previous Contents

Next: Precursors **Up:** An Introduction to Pattern **Previous:** Bibliography *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Linear Images **Up:** Syntactic Pattern Recognition **Previous:** Syntactic Pattern Recognition

Precursors

Syntactic Pattern Recognition was inaugurated by King Sun Fu, who wanted to extend some syntactic ideas to images. He tried to do it by the following approach: first he would take his image and scrutinise it carefully for bits of subimage that, more or less, recurred. If, for example, he had handprinted alphabetics, he would find bits of curves and lines from which he could regenerate the image, or at least an approximation to it, by gluing them together in some way. He assigned a symbol to each of his image primitives, and constrained the relationships between the primitives. In the simplest case it was just an order relation, and the image decomposed into a string of primitives and hence was described by a symbol string. Rafael Gonzalez has also written about this approach. Pavlidis has also considered the issue of structured objects, usually images.

The set of strings which you get from a family of images which are of the same type, for example handwritten words, or Kanji characters, where repetitions of the same word or character generate generally different strings which are nevertheless classified together, constitutes a language in the formal sense, indeed a stochastic language.

Now we produce a grammar for this language, usually a stochastic grammar. So we have one grammar for each of the families of image. Finally, for a new object, we reduce it to a string of primitives and calculate the probability for each grammar. The best bet wins. This is similar to straight statistical modelling of points in but gives us a class of models for languages instead. It should be noted that this is almost exactly what is done in the standard VQ/HMM automatic Speech Recognition systems. The space is chopped up into lumps, and it is the lump through which the trajectory goes which determines the symbol. The symbol strings are then used to generate a hidden markov model, a stochastic grammar.

There are four problems associated with this plan:

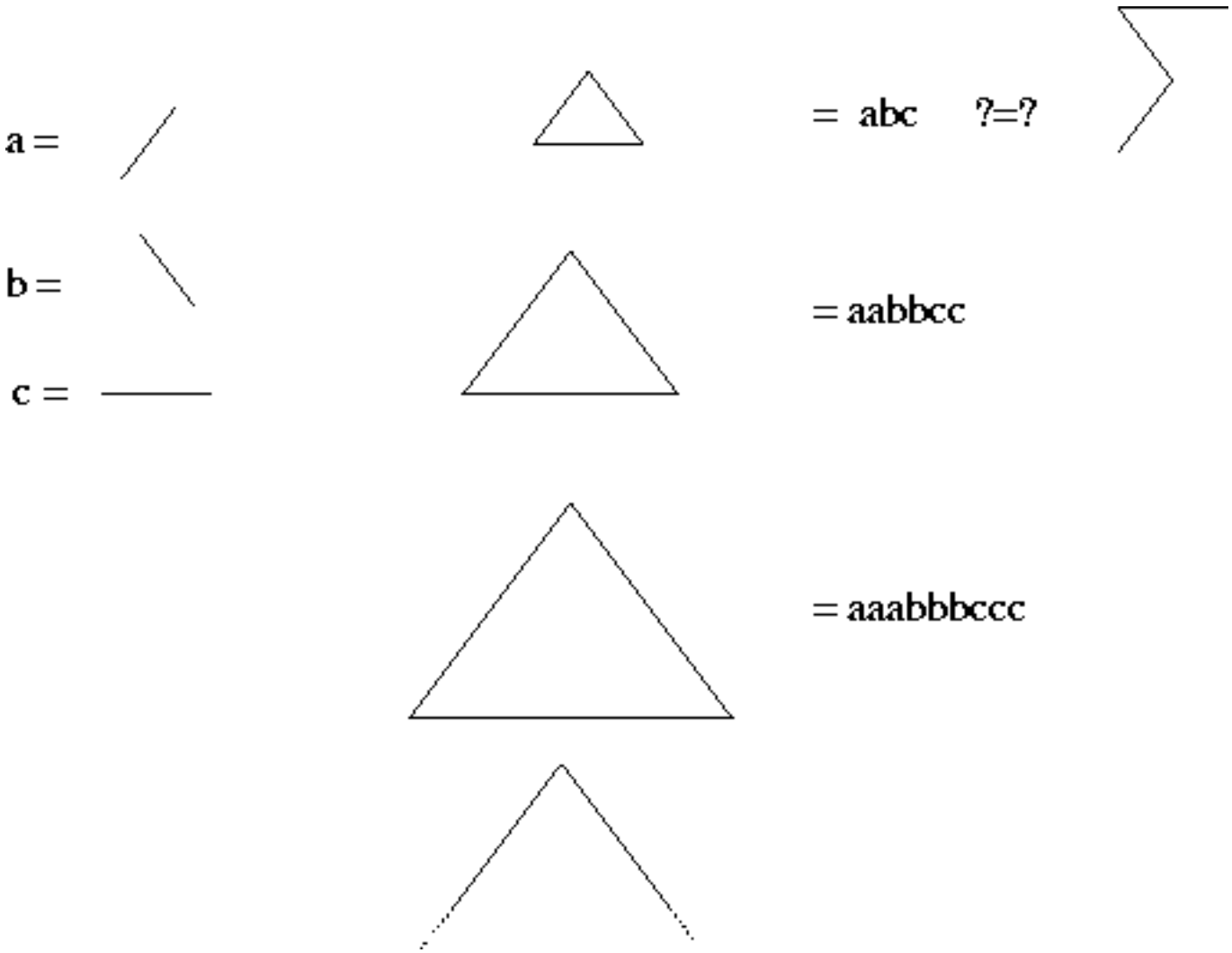
First, the order relation may not suffice for a specification of the way the primitives are related to each other, and generally does not.

Second, the system for choosing the primitives is frequently fraught with problems and normally involves human intervention. It should, one feels, be taken care of automatically.

Third, the system of matching the actual signal to the appropriate primitive may be equally doubtful and must inevitably entail some degree of forcing a continuum of possible forms into a discrete set of allowed descriptors. It is reasonable to want this to emerge from the data rather than to be forced upon it in essentially arbitrary ways.

Fourth and finally, the business of inferring a grammar from a sample of strings is often difficult and sometimes impossible to do in any manner which does not feel arbitrary and insanitary.

Figure 8.3: Classical Syntactic Decomposition.



To bring home all of them at once, suppose we feed the system a set of images of equilateral triangles, all scalar multiples by an integer of a smallest such one. We provide the system with primitives as shown in *fig*. 8.3. I have left arrows off, but we can take it that line *a* goes uphill, line *b* downhill, and line *c* from right to left. Then the string for triangles of side *n* units is $a^n b^n c^n$ which is a context sensitive language. If we constructed a hexagon by

$$a\bar{c}b\bar{a}c\bar{b}$$

where the bars denote the opposite direction, we can distinguish the objects which are hexagons from those which are triangles by examining the syntax of the strings. Triangles don't contain $\bar{x}$ for any *x*. Other strings such as $a^n c^n b^n$ are different triangles.

The system seems absurdly rigid, having no way of describing a triangle half as big again as the basic one. One could find ways around this, but what if the edges didn't quite meet? What if the lines were curved a little? It looks as though some sort of a symbolic strait-jacket is being forced onto a geometric

world. The straight-jacket approach leaves one wondering how one is going to decide to push cases into the formalism in general.

And finally, one might be a little unhappy about a system which had to do such careful counting in order to infer the language. Inferring regular languages is easy, but inferring context sensitive languages does not sound a good way to go.

If one contemplates this example and the fixes required to make it fly, one feels some sympathy for the basic insight: we need some mechanism for obtaining descriptions of at least some geometric objects in terms of levels of structure, just as we have with natural language. But it is less than clear that the program inaugurated by Fu is the way to go forward. It seems more natural to preserve the geometric structure of images for as long as possible, until the description at a symbol level occurs naturally.

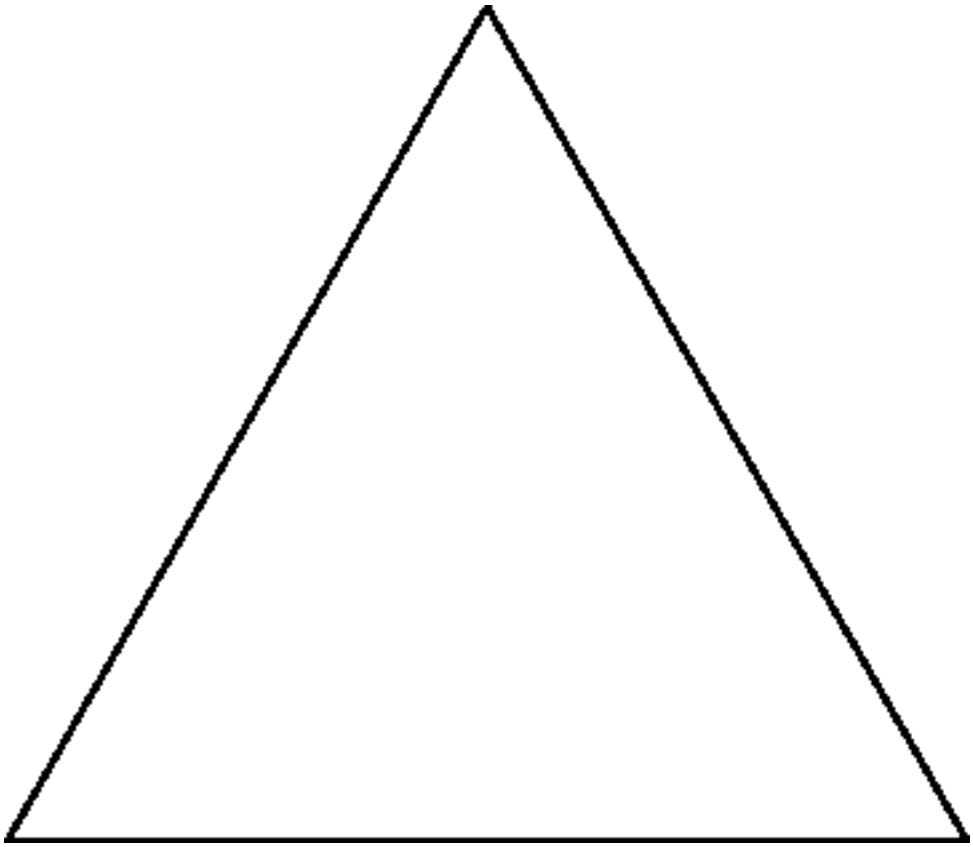
There are many other kinds of grammars which have been devised, in particular graph-grammars for graphs and stochastic grammars for images (Markov Random Fields) but none have been free of flaws. When they allowed inference, it did not lead to hierarchies of description which look the least like natural language structures. In what follows, I shall outline a theory which does it in a more agreeable manner. In order to avoid excursions into category theory, I leave the details of the abstraction to papers mentioned in the bibliography, and shall instead concentrate on giving algorithms which perform the needful operations. I start with some very simple cases in order to focus the mind on the essentials.

Next Up Previous Contents

Next: Linear Images **Up:** Syntactic Pattern Recognition **Previous:** Syntactic Pattern Recognition *Mike Alder*
9/19/1997

Linear Images

Figure 8.4: A pixel set with some structure.



In *fig*.8.4 we have a representation of a finite pixel set such as might be seen on the screen of a computer.

It is easy to see some structure in the image: it is far from being a random set of points inside the rectangle which is the computer screen.

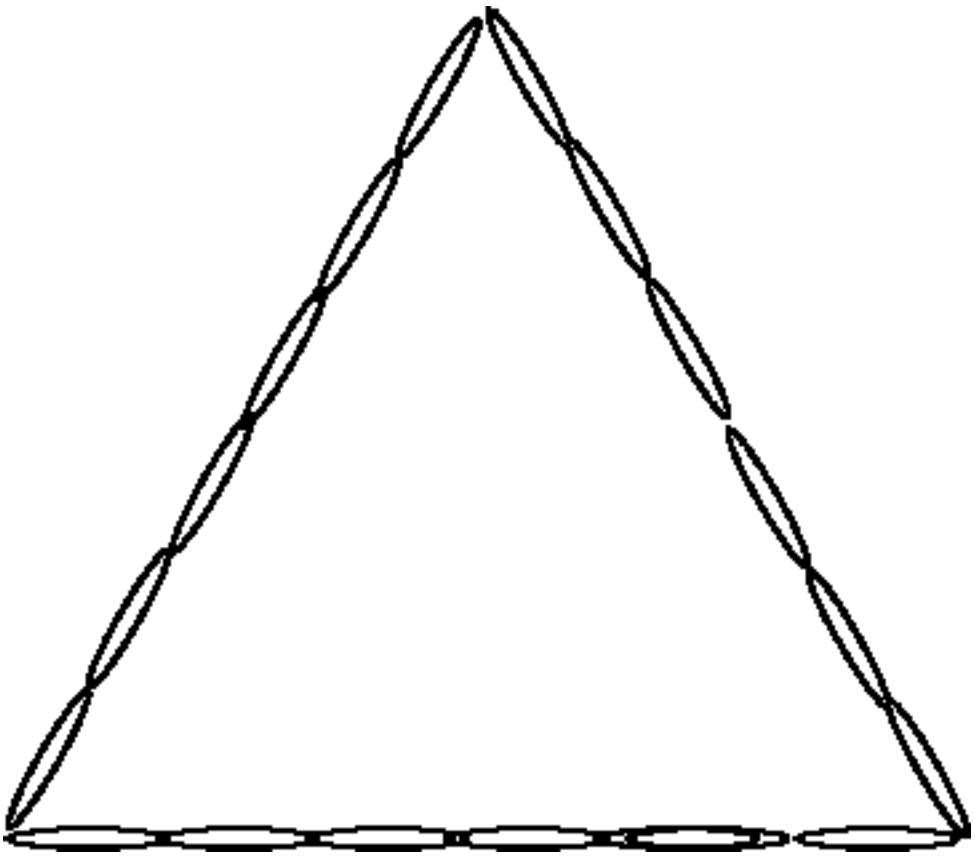
Let us first choose to regard this as a finite set of points in a bounded region of $\mathbb{R}^2$. This representation allows us to treat resolution issues as a separate matter. From now on I use the term *pixel* and *point of the image* interchangeably.

I propose to treat every point of the image as being a *symbol*, in an abstract sense which I shall clarify by example rather than by formal definitions. I want to apply to the image the same technique which chunks a stream of symbols (letters) into a new stream of higher level symbols (words) from a different alphabet.

I shall thus have a chunking UpWrite process which will take some subset of the image and declare it to be a single chunk.

I proceed in a manner essentially similar to the chunking process for letters. I choose a *resolution radius, k*, corresponding to the *k* in a choice of *k*grammar. I choose a pixel and set up a ball of radius *k*, which need not be an integer. I then take the set of all points of the image within the ball. This has to be turned into a local predictor of other points. There are many possibilities, but I shall simply compute low order moments for the set of points in the ball. For simplicity of exposition, I shall compute zeroth, first and second order moments of the set. This is equivalent to computing the number of pixels, the centroid of the set, and the covariance matrix of the set of points inside the ball. I may conveniently represent the quadratic form specified by the first and second order moments by drawing an ellipse, as in chapter four. This becomes a predictor of adjacent points, and also of adjacent ellipses, in an obvious way: we may expect points within some Mahalanobis distance of the centre, and we expect other balls to be placed on the image, so we expect other ellipses with centres about Mahalanobis distance two apart, most probably, therefore, along the major axis. I remove the point selected as my first centre or even all the points in the ball, and repeat with another choice of ball. In this way I can rapidly reduce the diagram to the set of quadratic forms shown in *fig*.8.5.

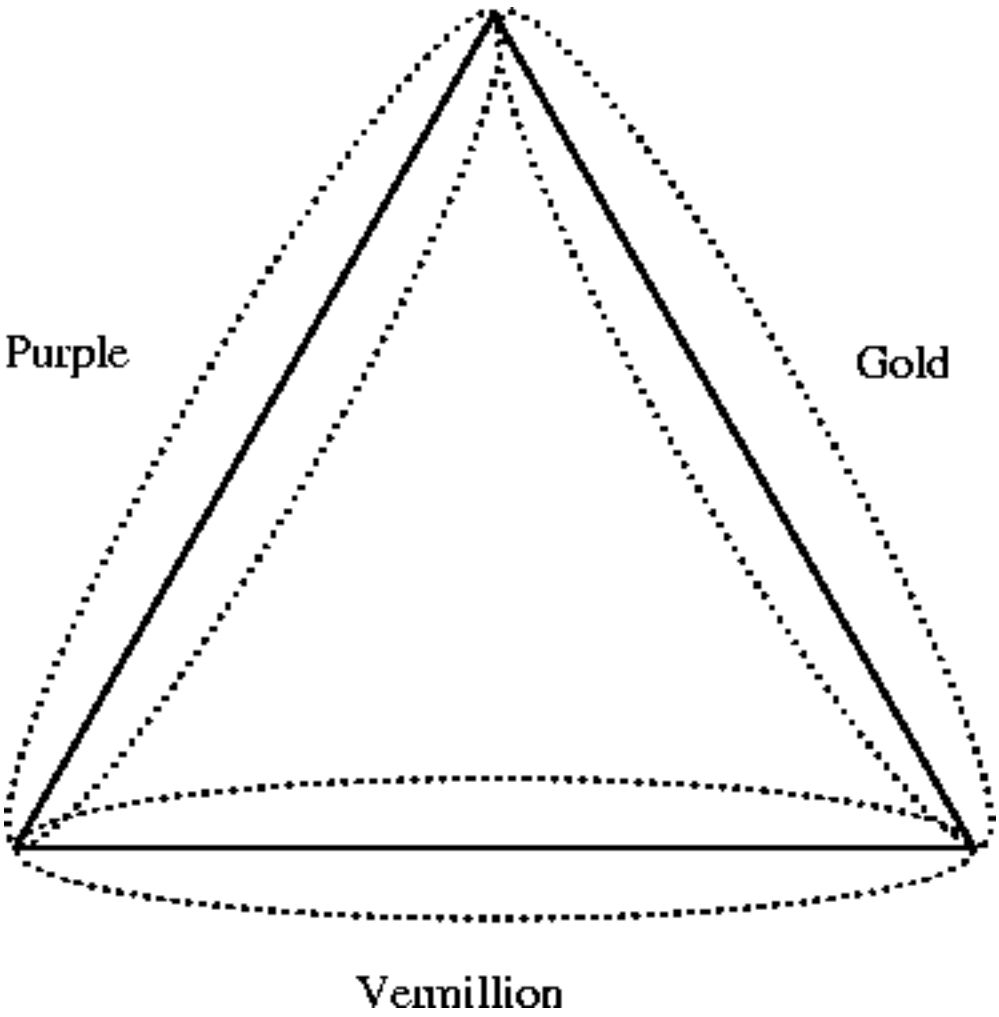
Figure 8.5: Prelude to Chunking.



We have now got something which is analogous to the *k*gram predictor, but which works in a wholly geometric setting. We use it to perform entropic chunking of the pixel set. More precisely, we `colour' a

set of pixels inside one ellipse, say, purple. If the pixels in a neighbouring ellipse are reasonably well predicted, i.e. if they are in the same line in roughly the expected place, we colour them purple as well. If not, we don't. We proceed in both possible directions, colouring pixels purple, until we come to an end where the behaviour is different. We then select another pixel, uncoloured, and another colour, vermillion say, and repeat. The effect of this on our triangle is clear enough: each side gets painted a different colour and we wind up with three colours. Each colour denotes a chunk of pixels.

Figure 8.6: Chunks.

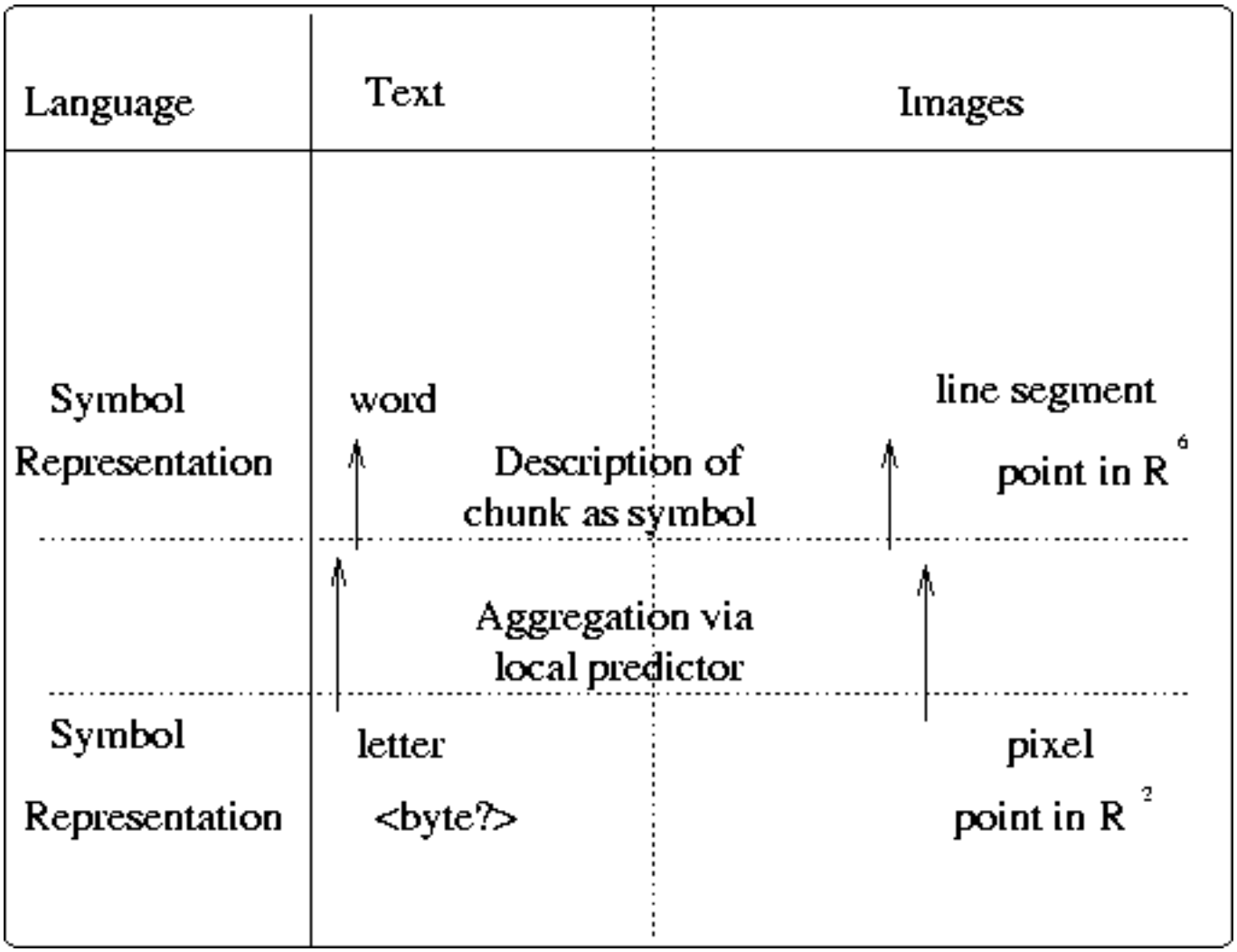


The final stage of the chunking UpWrite is to assign a symbol to each chunk from a new alphabet. Since we want geometric representations, we want a single vector which describes the set of points along an edge. Since we want it to work for other sets in other images, it must not be specific to line segments but must work for any shaped set of points . We choose, from many possible representations, to compute low order moments. For straight line segments, zeroth, first and second order suffice. So we simply compute three sets of moments, one for each side of the triangle, and since the original points are in $\mathbb{R}^2$, and there is one number for the zeroth order moment, two first order moments and three second order

moments, we wind up at the UpWrite stage with three points in $\mathbb{R}^6$ as a description of the triangle.

We have now UpWritten a set of symbols (points in $\mathbb{R}^2$) to a smaller set of symbols from a different alphabet (points in $\mathbb{R}^6$) in a way which is analogous to the way in which we extract words from strings of letters. The situation can be described by the diagram of *fig*.8.7:

Figure 8.7: A useful Analogy.



Having done it once, there is nothing to stop us doing it again. In this case a judicious choice of radius will include all three points, and there is only one chunk. The whole lot may be UpWritten to a set of moments. In this case, again the zeroth, first and second order moments suffice to specify the set of three points. We obtain one point in $\mathbb{R}^{28}$ describing the triangle, at the second level of UpWrite.

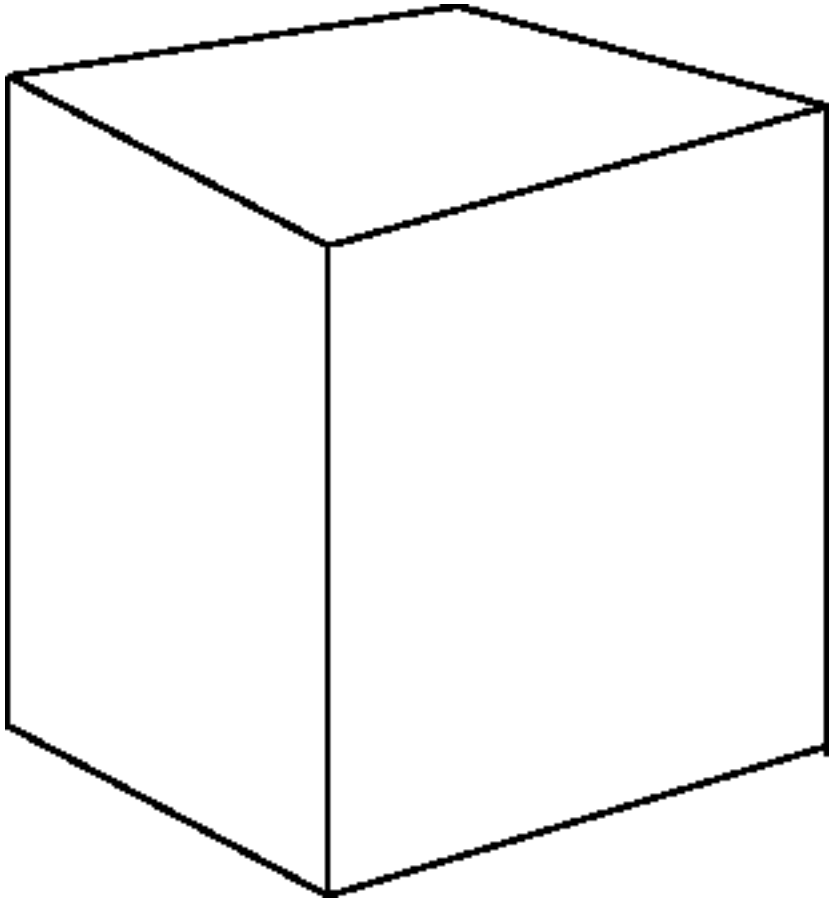
Having done this with one triangle, we may do it with another which is smaller or bigger than the original one. Suppose we take a scaling parameter, *r*, and multiply the original image by *r* in order to get

the scaled version. I shall suppose that there is no translation of the images. Let us suppose, in a spirit of charity, that r varies from about 0.5 to 1.5. Now provided that the resolution radius is big enough to enclose enough pixels to avoid total degeneracy in the covariance calculation, and sufficiently small not to enclose more than one edge at once, the same process will work to the first level. Each edge will be in a different location and will be scaled appropriately. At the second level of UpWrite, the centroid will be in the same position, but the covariance will go up as the square of the scaling term. Thus the different triangles will UpWrite to points lying along a smooth curve in the topmost space, $\mathbb{R}^{28}$ in this case.

This set of `triangles', can also be described by moments (although greater than second order might be expedient) to specify it as a point in yet another space. This point in the top level space corresponds to the language of *fig*.8.3. From it, we can generate the points of the triangle space. From each of these points we can DownWrite to a set of line segments, and from each of these to a set of pixels. The DownWrite is not generally unique, but DownWrites often aren't.

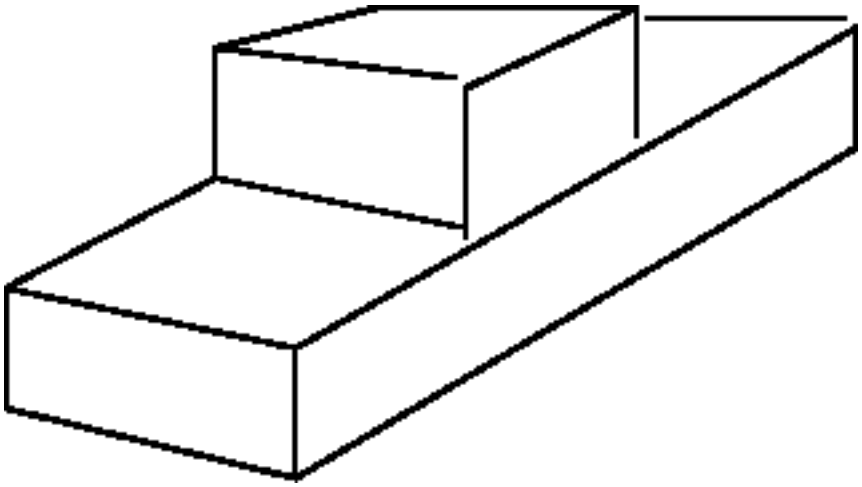
It should be clear that we have all the capacity to precribe equilateral triangles of the Fu program, and rather a lot more besides. If someone gives us a collection of triangles and a collection of squares, we will get two clusters in the space formerly containing only triangles as points. Providing we don't scale down to zero, the clusters will be disjoint. The clusters will be named as vectors at the top level, but because they are quite separate and no intermediate forms occur, they might just as well be described by symbols. Indeed, a symbol may now be interpreted as being a point in $\mathbb{R}^{n}$, assigning an alphabet of them a metric structure.

Figure 8.8: An object requiring more levels for its decomposition.



If instead of having a triangle we had presented the system with a drawing of a cube, as in *fig*.8.8 we should have been obliged, in order to do a good job of capturing the higher order structure, to go to a higher level of UpWrite. And of course, it can go further, as the world of block objects shows.

Figure 8.9: And one requiring even more.



It is possible to write a program then, which can be fed as input a set of line segments at various

orientations and locations. If we had given it a set of vertical short lines, and a set of horizontal long lines, disposed anywhere on the screen, the program would produce two clusters in $\mathbb{R}^6$. Given a label to each cluster and a new line segment, we could use standard methods to decide to which family it belonged. Moreover, it is a simple extension to the program to have it presented with squares and triangles, so that having first found the edges, it then UpWrites each square to four points in $\mathbb{R}^6$, each triangle likewise to three points in $\mathbb{R}^6$, and then performs exactly the same trick again to obtain a cluster of square points and a cluster of triangle points. These might be taken to be in $\mathbb{R}^{28}$. This would give us a natural system for classifying triangles and squares. It is easy to see that the same process can be applied again to higher level objects such as cubes and pyramids, and even higher order objects built up out of boxes. And the only limitations on this process are computational.

Next Up Previous Contents

Next: Curved Elements **Up:** Syntactic Pattern Recognition **Previous:** Precursors *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Parameter Regimes **Up:** Syntactic Pattern Recognition **Previous:** Linear Images

Curved Elements

The images containing only blocks, and things made from blocks, are but a small and unnatural subset of the set of all images. It is essential to be able to treat of more general objects before we can feel warranted in taking ourselves seriously. So the question arises, can we decompose more complicated objects than linear images? Happily the answer is that we can. If we have an image built up out of curves, at some suitable level of resolution, between where the noise intrudes and the global structure intrudes, we may use the same process to perform chunking, and the same process to perform the UpWrite of each chunk. Higher order moments than two will generally be necessary in order for the general shape of the curved chunk to be described with sufficient precision to allow a convincing DownWrite. There is plainly, however, no intrinsic obstruction to dealing with images composed of piecewise curved pixel segments. Moreover, the resolution radius is a parameter which can be adjusted in some cases to distinguish noise from signal, a point which will be enlarged on later.

The significance of curved elements is, as the thoughtful reader will have noted, that they arise rather naturally from physical objects coming in front of other physical objects. Boundaries can often be taken to be piecewise curved objects with noise.

There are various ways of chunking into curves: the program *recog* which can be found on the site:

`ftp://ciips.ee.uwa.edu.au/pub/syntactic/c_code/rht/`
shows a variety of methods.

Next Up Previous Contents

Next: Parameter Regimes **Up:** Syntactic Pattern Recognition **Previous:** Linear Images *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Invariance: Classifying Transformations **Up:** Syntactic Pattern Recognition **Previous:** Curved Elements

Parameter Regimes

There are two parameters associated with the chunking UpWrite as described so far, one is the resolution radius and the other is the order of the moments used to compute the UpWrite symbol assigned to each chunk. It is natural to wonder how one can pick the best values for these parameters.

A choice of resolution radius amounts to a decision that smaller scale variation will be treated as noise and averaged over. The decision as to what constitutes noise and what signal is seldom clear cut in the real world.

The problem is not well-posed: in order to decide what is the optimum resolution one must know what one is trying to perceive in the image, and there may be structure at many levels. The following example was suggested to me by Richard Thomas of the University of Western Australia:

Figure 8.10: A bit of a page of handwritten text.

Look at a page of handwritten text, as in *fig.* 8.10.

If we choose a resolution radius bigger than the mean distance between lines of text, we get an image, after chunking, which looks like the region in *fig.* 8.11.

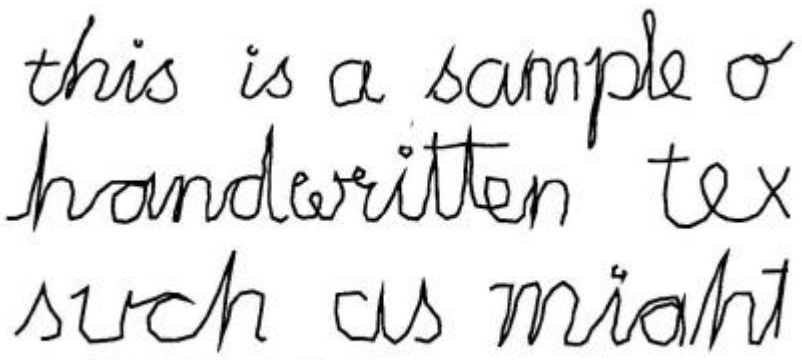
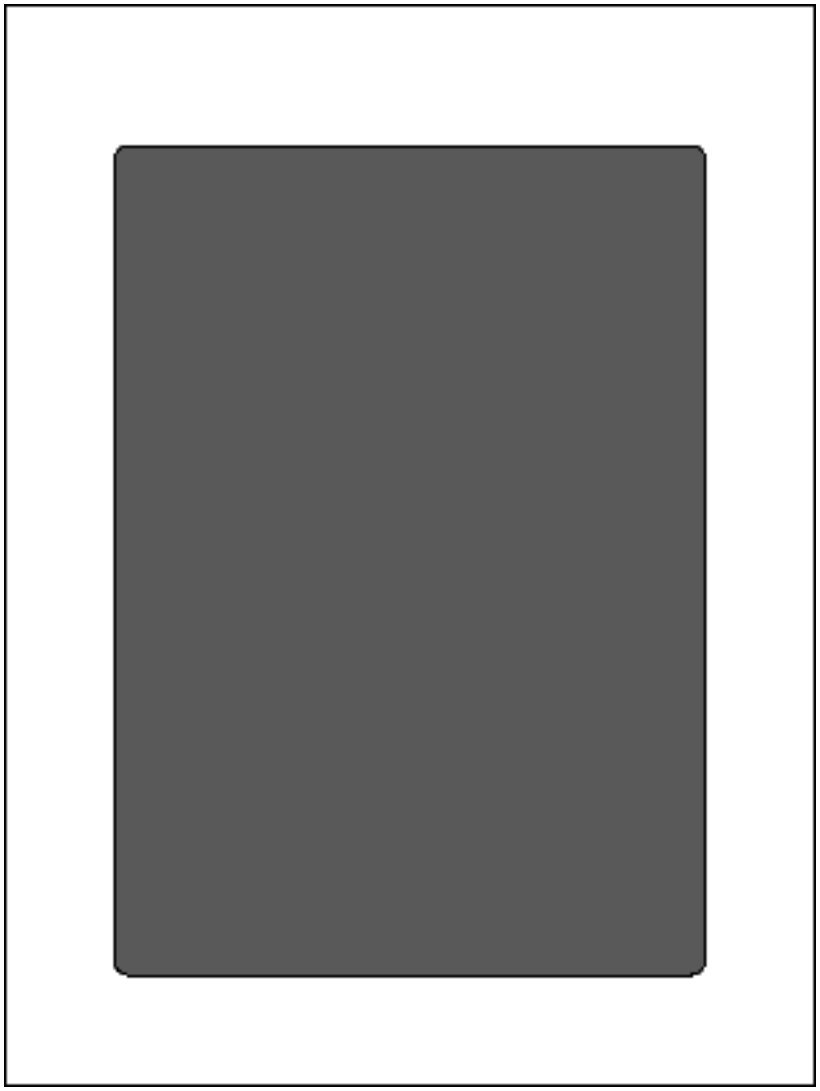
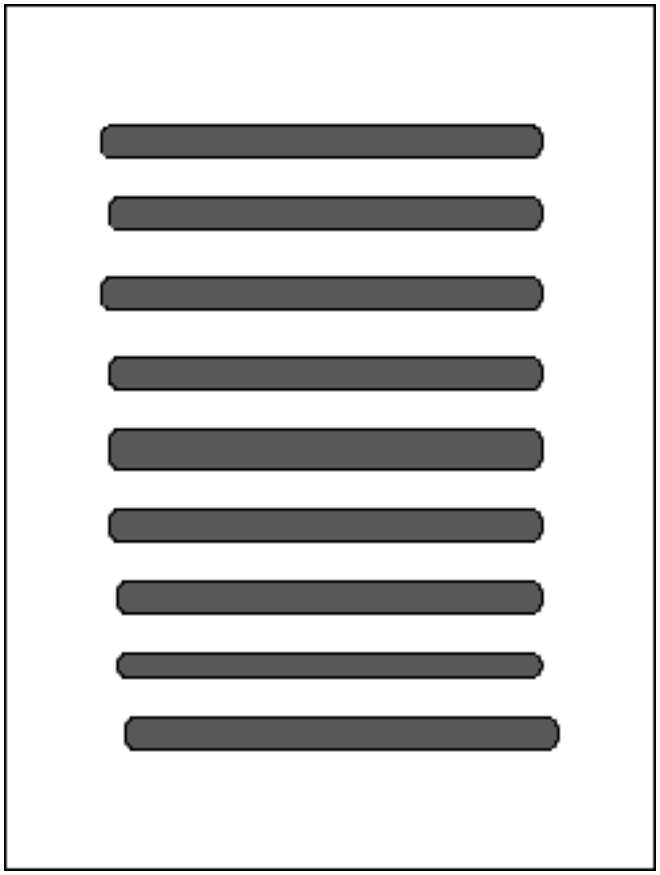


Figure 8.10: A bit of a page of handwritten text.



If we had used a smaller radius, we should have still got the same outcome, until the resolution falls below a certain value, whereupon we should have got *fig.* 8.12 briefly, before getting more details of the word level.

Figure 8.12: The page at intermediate resolution.



It is clear that there are stable regimes for the parameter values, and that between them there are fairly fast transitions. It depends rather on what we are trying to do which of these regimes matter to us most.

I shall often observe that a problem is solvable by the general means outlined in this chapter, by simply noting that there are stable parameter regimes which solve them. A purist might grumble that I do not give procedures for finding them which are practicable on present day computing equipment, and that investigating a large sample of parameter values concurrently requires specialised hardware of colossal power. This is correct. In a later section I shall discuss what we are here regarding simply as a more sophisticated pattern recognition system, in the different and significant context of a new class of neural models, and the suggestion will be made that neural implementations would proceed at many different scales or parameter values concurrently, and would also infer the regime structure, and correlations between regimes.

In particular applications, we may cheat and use the human eye to decide on a suitable scale value. This is generally more convenient than trying a range of values, when it is possible to do it at all.

The situation with respect to order of UpWrite is, I suspect somewhat different, but I shall go into that below when I discuss neural implementations of the UpWrite process.

Next Up Previous Contents

Next: Invariance: Classifying Transformations **Up:** Syntactic Pattern Recognition **Previous:** Curved Elements *Mike Alder*
9/19/1997

Invariance: Classifying Transformations

An important case of families of images occurs where there is essentially one object, which generates a set of images by the application of a Lie group of transformations, or some neighbourhood of the identity in such a group.

In order to focus ideas, I shall consider again the simple case of triangles which are all equilateral and which differ only in the scale. I shall suppose that we have a standard such triangle with the centroid at the origin, and that by taking a range of scalars between 0.5 and 1.5 I have shrunk or enlarged the basic triangle to give a set of images, one for each scalar.

If we UpWrite the triangles to points in $\mathbb{R}^{28}$ in the standard way, we see that the set will constitute an embedding of the interval [0.5,1.5] in $\mathbb{R}^{28}$. The line will be curved rather than straight, and in practice we will get a finite point set on or close to the idealised embedding.

Doing the same thing with squares and hexagons will yield two more such curves. If the size of the standard object in all three cases is similar, then the curves will lie approximately on a space of `scaled planar objects'. Other objects, such as pentagons, heptagons, octagons and circles will also lie on such a space. This space is foliated by the curves representing the Lie group action. Given a new object, it is possible in principle to estimate the scaling curve upon which it lies, and to be able to recognise a scaled version of the object which has never been seen before. In other words, we may learn the transformation.

Less ambitious undertakings have been found to work quite satisfactorily: in an application involving the recognition of aircraft sillhouettes, the space of each curve was found for two aircraft, and a hyperplane fitted to each of them. Scalings from 60% to 140% of the original size were used. A new silhouette of one of the aircraft was then presented to the system, and it was classified according to which of the two hyperplanes was closest. By this means it was found possible to obtain correct classification down to about 20% of the original size, at which point resolution issues came into play. The system was capable of learning the space and performing creditable extrapolations in order to achieve a classification.

Other Lie Groups occurring naturally are the group of shifts or translations, and the group of rotations. The cartesian product of these spaces with a scaling space gives, for each planar object, a four dimensional manifold embedded in the top level space.

It is of course also possible to use invariant moments, but this presupposes that we know the appropriate group. If we contemplate the general situation, we see that this may be infeasible. For example, an image of a bar code printed on a can of beans is deformed in a way which is rather harder to specify, and the transformations to a trajectory in the speech space involved in (a) breathing over the microphone, (b) enlarging the vocal tract, (c) changing sex, we see that although we have group actions on the data, and although factoring out these group actions is highly desirable, an analytic description of the action is at least difficult and generally impossible to obtain.

A symmetric object will not generally give an embedding, but will give an *immersion* of the group. If, for example, we rotate a square, the result of UpWriting will not embed the circle group SO(2) in the top level space, because the path traced out will recur after one quarter of a circuit. We get a circle mapped into the space, but we trace it out four times. Given noise, quantization and finite sampling, the result may be four very close lobes, each close to what is topologically a circle. This gives us a means of identifying symmetries in the object.

If we consider three dimensional rotations of an object such as an aircraft which is then projected onto a plane, as when we take a video image, we get, via the UpWrite process, a space which may not be a manifold, but which is likely to be one for almost all of the points. If the reader thinks of something like a circle with a diameter included, he will observe that the object is pretty much one dimensional, in that for most points, a sufficiently small neighbourhood of the point will give a set which looks like an interval in $\mathbb{R}$. The two points where the diameter meets the circle are singular, but there are only two of them. For most points of the UpWritten image, a small perturbation derived from a rotation in three dimensions, will give another point which if DownWritten to the image set will give something qualitatively the same. Imagine an aeroplane or a cube `in general position'. For some views of the cube or the aeroplane however, the image degenerates into something different from neighbouring views, as for example when the cube is seen along an axis orthogonal to a face and we see a square. There may be discontinuities in the UpWrite at such locations; nevertheless, the space is mainly a manifold, it may be UpWritten as an entity in its own right. When this is done with distinct classes of objects, we recover at this level separated objects which may conveniently have the symbol designating them replaced by an alphabetic term, since the toplogy has become trivial. Thus we see that the case of strings and symbols in the usual sense may be recovered as a degenerate case of the more general topological one.

Next Up Previous Contents

Next: Intrinsic and Extrisic Chunking **Up:** Syntactic Pattern Recognition **Previous:** Parameter Regimes

Mike Alder
9/19/1997

Intrinsic and Extrisic Chunking (Binding)

I was guilty of glossing over a point in drawing an analogy between *k*gram predictors in natural language text, and the use of forms or gaussian distributions to predict close pixels in an image. The point merits some attention. In the case of the *k*gram predictor, we actually do not simply use one *k*gram to predict the next symbol, we run along a lot of text and generate a probabilty distribution conditioned by the preceding (*k*-1)gram. Thus we apply a knowledge of what other *k*grams consist of. In the case of the forms on arrays of pixels (or higher level objects), we used only the distribution of points within one region. I shall call the latter *intrinsic* and the former *extrinsic* chunking.

It is possible to perform both in each category, the text or the images. If, for example, we used *k*grams on a text consisting of long strings of one letter followed by long strings of another, then in the middle of one string we should predict another letter the same, and the chunks (words) would all be strings of one letter. Dually, if we took sequences of forms and learnt that one form tended to predict another in a particular position reasonably close to it and in a similar orientation, we should be able to use extrinsic chunking of forms. We see also that storing the possibilities and their counts allows us to extract straight line segments and also `corners' as significant `features', by what is a *clustering UpWrite* instead of the chunking UpWrite we have used exclusively thus far in our discussion of images. Here, we do not use the entropy of the conditional probability distribution, we use the information supplied to the predictor by the data.

This is of particular significance when we contemplate alternative training procedures for generating recognition of such things as cubes. It was tacitly supposed that the system was trained first on line segments, then on faces consisting of parallelograms, and finally on cubes, so that it would have some regions of the UpWrite spaces at intervening levels already containing points which could be recognised. It is hardly feasible however to train a human face recogniser on noses, eyes and mouths separately, and we confront some problems when we try to train exclusively on cubes. The question is, what particular set of points representing line segments should be selected so as to find faces? Why not select the `Y' shaped triple of line segments which are a distinctive feature of a cube seen in general position instead? Indeed, why not? It seems likely that such `features' are indeed used by the human eye.

In order to extract a face of a cube, as a distinctive feature (i.e. as a point in a suitable UpWrite space), in a set of images of cubes, we would seem to need to perform a rather painful combinatorial search through the space of line segments, choosing some subset which recur often enough to be worth grouping into an entity or feature. This problem of entity formation has troubled psychologists and also neurophysiologists, by whom it is referred to as *binding*. It is an important issue, and I shall try to persuade the reader that the method outlined here reduces the matter to a formal problem in geometry and statistics.

Suppose our data consists entirely of binary images of linedrawings of cubes, each in general position, differing in respect of location, size, and also orientation with respect to the plane on which they are

projected. Suppose, moreover, that each image is perfect, with only quantisation noise occurring. Then there is nothing to be gained from a syntactic decomposition at all, and it would be sensible to chunk the whole object on each occasion into one set of pixels, and to UpWrite this using higher order moments to obtain an embedding of the transformation group. The UpWrite would be all done in one step, and would hardly be justified in being called an UpWrite at all. The presumption is that new data will also look like a cube in general position, so there's nothing to classify. Similarly, if there are images of pyramids in general position *and* images of cubes in general position, then we could simply UpWrite each directly to a point in a moment space and obtain two manifolds, one for each object, with little difficulty. Recognition could be accomplished with another level of UpWrite or in other words by modelling each manifold crudely and computing distances.

Now let us suppose that the data set of cubes and pyramid line drawings is modified to contain noise at the level of the line segments, that is to say, some of the lines are too short, too long, slightly slanted, or slightly displaced from the ideal position. Now a direct moment description of the sets leads to very considerable variation at the UpWritten level. If, in addition, some of the lines are missing altogether, something that the human eye will recognise as a cube with a bit missing will not usually be correctly classified. The situation arises naturally in the case of handprinted characters, where variation at the stroke level can reduce recognition rates considerably.

In this case, there is a plain evolutionary advantage in finding intermediate structure, and it can be done by intrinsic chunking, or by extrinsic chunking. The former is simpler, but the latter is more powerful, since it tells a program trained on the data described, that at the lowest level of pixels, there is a structure which can be described locally, and which extends either linearly or so as to produce corners. We collect sets of quadratic forms and discover that they are locally correlated in one or the other of two simple ways. In general, we take a small neighbourhood of a point in the space, and describe it by UpWriting it. Now we take a ball in the UpWrite space and UpWrite the points in that. Now we look for clusters in this space. In the case of the cubes and pyramids, we discover that in the case where we are looking at, say, pairs or triples of ellipses, each describing a small region of the image, we get a cluster of ellipses in straight lines. We may get a smaller cluster of ellipses in right angled bends for cube images, and similarly for 120^o bends if there are pyramids. The existence of the clusters tells us that there is structure and that it should be exploited. It also tells us how to exploit it.

We select then a prediction system extracted from the clustering, and use it to predict where the next ellipse should be in each image, and of course use the information supplied by the image to partition the pixels into line segments.

This procedure works at every level. If there is noise at the level of faces of a cube or pyramid, we look at the sets of points in $\mathbb{R}^6$ corresponding to, say, a cube. There will be nine of them. Taking any neighbourhood of one big enough to include at least one other such point, and taking note of where it is by computing moments of the pair, we repeat on a lot of images of cubes and discover a cluster. This cluster might denote the angled bends between adjacent edges, the `Y' shaped feature in the middle, or the parallelism of opposite faces. All that is necessary is that the property consitute a cluster when data is accumulated from a set of images, in other words that it recur with sufficient frequency. If the neighbourhoods chosen are small, then we may find ourselves going through intervening stages where we group together edges into corners or opposite faces. If they are larger we shall find faces, and if even larger, we shall find only the whole cube.

It is plain that there is a case for some kind of feedback from one level to its predecessor saying what is a sensible size of resolution radius. If the data is too surprising, maybe a smaller choice of radius at a preceding level would be a good idea.

Note that the combinatorial issue of which points to group together is solved by the following strategy:

- take a point and some size neighbourhood in the space.
- Describe the shape of the cluster of points in the neighbourhood in an UpWrite space.
- Repeat for lots of different points in the same set and also in different data sets.
- Look for clusters. If the clustering is poor, change the size of the neighbourhood in the first step and try again, or if you have a brain on which to run things, do lots of different choices in parallel and select out all the cases where there is clustering.

Such an approach has been used in finding strokes making up the boundaries of moving solid objects such as heads.

Next Up Previous Contents

Next: Backtrack **Up:** Syntactic Pattern Recognition **Previous:** Invariance: Classifying Transformations

Mike Alder
9/19/1997

Next Up Previous Contents

Next: Occlusion and other metric **Up:** Syntactic Pattern Recognition **Previous:** Intrinsic and Extrisic Chunking

Backtrack

This chapter started with the observation that there is a fundamental issue of segmentation, which should properly be part of the classification process: a little thought shows that we have indeed the possibility of incorporating the two matters in a coherent way. If we proceed to extract structure from a postcode which has the stroke of the five distinct from the body, then we can easily see that there will be a stroke cluster corresponding to the digit `5' provided only that there have been lots of 5's offered to the system. Moreover, cases where the connection is made and where it is not quite closed will simply be variants of the same cluster of three strokes. Since the stroke triple consisting of all the strokes for the digit `5' will occur with relatively high frequency, we will get a cluster of these triples.

In effect then, the system under discussion will do automatically the aggregating into what sets should become segments, hierarchically, under the accumulation of data. We have disposed of the segmentation problem.

There were two simple data sets which all existing classification systems get wrong. It was asked if we could find a generic `fix' which would do better than the naive clustering methods discussed thus far under the heading of static pattern classification.

The answer is gratifyingly positive. Consider first the chess-board pattern. At the lowest level of aggregation, we find neighbourhoods in a small ball correlate with other adjacent small balls provided the diameter of each ball is small compared with the size of a square. The points are technically in $\mathbb{R}^3$, with the last component either +1 or -1 depending on the colour or type of point. The entropic chunking will thus collect all the points of a square together for the next level of UpWrite.

The next level now represents the alternating pattern. In a suitable space of square regions, perhaps crudely represented by fourth order moments of a square in $\mathbb{R}^2$, we get alternating blocks of points in the base level. Taking any one such point and looking at a neighbourhood, we get, as we go across the image, very tight clustering: we keep getting a point which is a cross being surrounded by opposite category points along ranks and files, and diagonals which are of the same type if the neighbourhood is slightly bigger. This structure is in effect learnt from other parts of the same data set. Using such a neighbourhood descriptor, the entire image is decomposed into iterates of this primitive. The system learns the process by which it appears to have been generated in the same way as the bag structure ABCDABCDABCD was learnt in the last chapter.

Classifying a point in a region is done by downwriting from the higher level description which will fill the empty square with points which although not present in fact are deduced from the higher level structure. Now we can determine the category of the unknown point by looking at its neighbours- not the actual neighbours but the virtual neighbours generated by the DownWrite. This is analogous to

identifying the word *profes5or* as being close to the word *professor* which has been seen before, DownWriting the word to its spelling (a rather trivial operation), and then identifying the symbol between *s* and *o*

We are using the same machinery in both cases, just doing it with different interpretations of the terms `symbol' and `concatenate' in fact. This point requires some mathematical sophistication not required of the reader of this book.

In the same way, the double spiral data set is a set of points in $\mathbb{R}^3$, the first two coordinates telling you where the data is and the last digit telling you if it is a nought or a cross. The local structure is quite different from the case of the chess-board pattern, but the same machinery works. The set of noughts is locally described by a set of ellipses, and local groups of, say three ellipses may be represented by giving their location relatively, or this information may be inferred from the canonical UpWrite via moments of the points in $\mathbb{R}^6$. The relationship will in fact change slightly, but predictably, as the data gets further from the centre. Extrapolation is therefore possible. This is done automatically by simply inferring the structure `rule' and DownWriting to get the distribution of points which the top level coding will generate. Again, the classification is accomplished by the same machinery, we look at the virtual neighbouring data obtained from the DownWrite.

Next Up Previous Contents

Next: Occlusion and other metric **Up:** Syntactic Pattern Recognition **Previous:** Intrinsic and Extrisic Chunking *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Neural Modelling **Up:** Syntactic Pattern Recognition **Previous:** Backtrack

Occlusion and other metric matters

An important part of the problem of binding or chunking is the following: Suppose I have seen lots of squares and triangles separately, and I see three sides of a square: how do I arrange to look harder for the fourth side? After all, I should have some expectation that there is a fourth side, so would you, and so, one suspects would a moderately bright cockroach. Does this emerge from the formalism? How does one deal with the situation whereby we have two objects, one partially in front of the other and we recognise them both? This is, if you reflect upon it, a clever trick.

The case of three sides of a square where we have formerly seen all four will do to start with, since the general case of missing data is essentially the same. It will by now be apparent that these issues do not make sense except relative to a body of data, so we shall suppose that many squares and triangles of different sizes and orientations and locations have been provided to the system, and also a similar set of triangles. Now we present the system with a three sided square, that is, one with a single edge absent. The system has to decide if this is more like a square or a triangle. If it concludes that it is a square, it has to supply the missing side. A related problem is that we give the system a noisy square and require the system to clean it up.

Now the UpWrite of the three-sided square will have a zeroth component saying it has three elements in the set, a centroid consisting of six numbers, and a covariance matrix of 21 numbers. This supposes that we have a resolution radius big enough to encompass all three edges. The six centroid entries will contain first the average of the number of pixels in each edge; these numbers will be very close, squares being what thery are. The corresponding variance entry will therefore be small and the centroid entry will be the same as the length of each side in pixels. The next entry in the centroid wil be the centroid of the centres of the edges. This will not be in the centre of the square because one side is missing. The centroids of the covariance matrix terms in $\mathbb{R}^6$ will likewise be distorted from what we might have expected had the fourth term been added in. Likewise, the covariance terms will be affected by the missing edge.

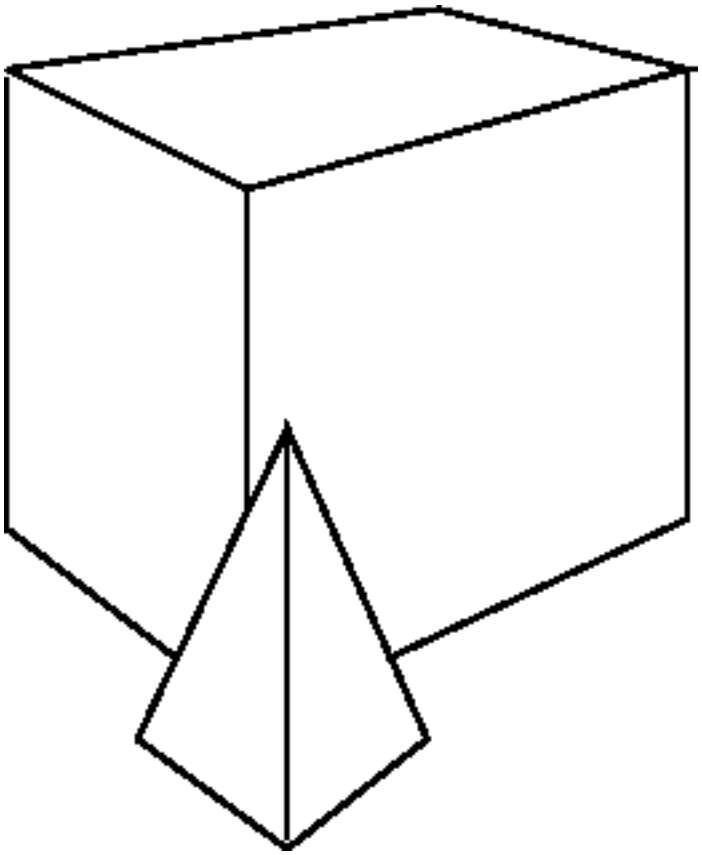
If we look at the (four dimensional) manifold of squares in $\mathbb{R}^{28}$, the six dimensional manifold of all triangles in the same space, and the single point representing the UpWrite of the three-sided square, we can ask, to which manifold is the new point closest? And with this comes the associated question, how do we measure distances? This is clearly vital here: if we give high weight to the first component, we observe that all triangles have three sides, this object has three sides, therefore this object is a triangle. If we giver high weight to the centroids as well, we shall conclude that the triangle it `is' must be equilateral. But if we give high weight to the covariance terms, we might complete the square instead of bending two sides to join the remote ends. Similarly, if we had used smaller neighbourhoods to get more levels of UpWrite, we should have been heavily influenced by the fact that we had two right angles in the three-sided square, and that right angles correlate with squares and not triangles. Which is the right way

to go?

It depends on the data. There is no answer to the dilemma of which is right, either might be reasonable, it would depend largely on what kind of data has been seen before. If there were cases of triangles which had gaps in them at one or more vertices, the case for thinking it a deformed triangle would be strengthened. If there were squares with holes in one or more of the sides, the case for a square would be strengthened. In general there is no way to tell, and both are possible.

To see how to use the extra data, say lots of squares with bits missing, note that the manifold of squares is now made rather blurred along the outlines by the noisy data of partial squares. Local descriptions of the structure of this noisy manifold will give, to second order, local quadratic forms near the centrally fitting manifold. We may use these to obtain an estimate of the metric in the region of the three-sided square. If deformations of this sort occur commonly, then the quadratic forms will have large extensions in the appropriate direction. We can think of them as giving estimates of the Riemannian Metric Tensor for a suitable metric. This solves, at least in principle, the issue of the best completion of the three-sided square. All we have to do is to find the closest and DownWrite it. Refinements using different levels of the UpWrite will occur to the reflective reader. It will be immediately apparent to such a reader, that excluding adventitious line segments to an existing square is essentially the same problem.

Figure 8.13: A simple occlusion problem.



The occlusion problem extends this quite directly. If we have a pyramid occluding part of a cube,as in

fig. 8.13 first we find the pyramid.

This is a matter of deciding to exclude some elements in a neighbourhood, as referred to dismissively in the last sentence of the preceding paragraph. (It may be a little longer to compute it than to describe it.) Next we remove the pixels belonging to the pyramid which we have by this time classified. We then take the object which is left and classify that. We discover that it may be completed into a cube by consistent and relatively small extensions at lower levels. This means we have to use the appropriate metric at the topmost level. The metric is learnt, either intrinsically from other parts of the same image or extrinsically by comparison with other images. Alternatively, we use a hierarchical probabilistic model to generate relative likelihoods of the two models given the data.

Next Up Previous Contents

Next: Neural Modelling **Up:** Syntactic Pattern Recognition **Previous:** Backtrack *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Self-Tuning Neurons **Up:** Syntactic Pattern Recognition **Previous:** Occlusion and other metric

Neural Modelling

- Self-Tuning Neurons
- Geometry and Dynamics
- Extensions to Higher Order Statistics
- Layering

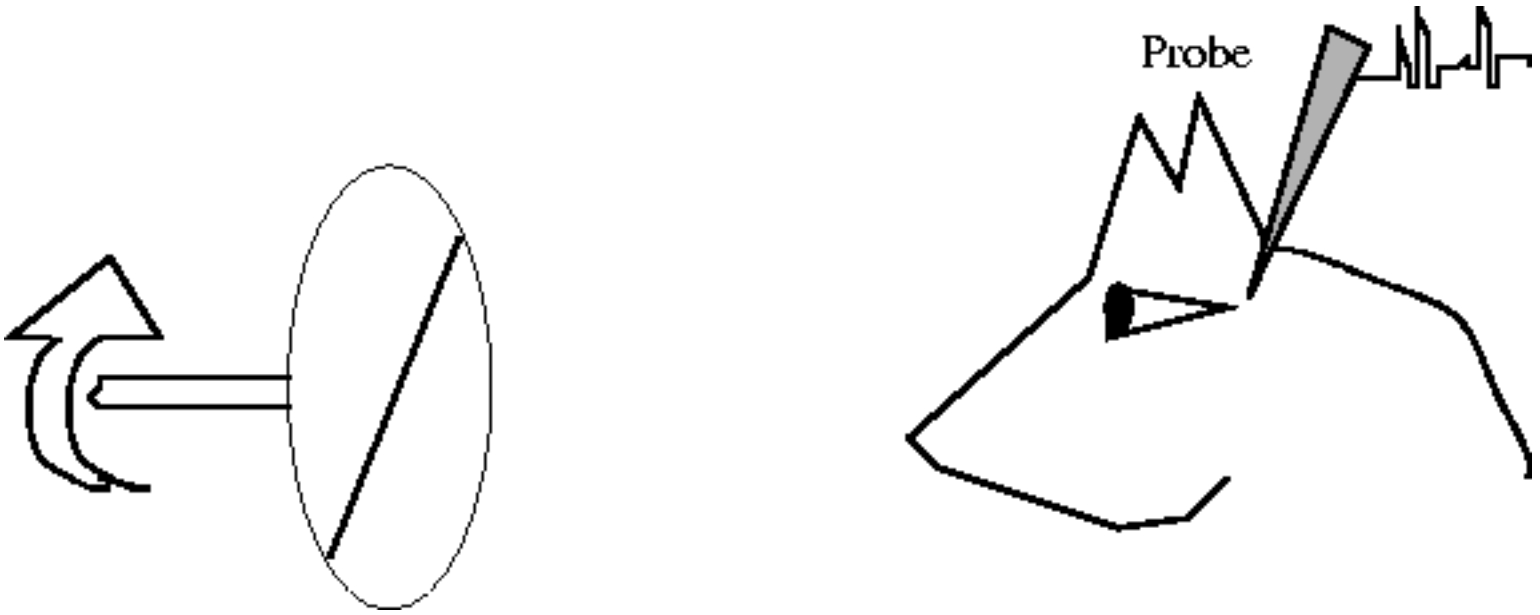
Mike Alder
9/19/1997

Self-Tuning Neurons

There is an interesting and suggestive experiment which has been carried out by Hubel and Wiesel, Hirsch and Spinelli, and extended by Blakemore. See the Bibliography for some sources, particularly the Blakemore paper. In the experiment, a cat has its head clamped and is kept fully conscious looking at a white disk free to rotate about an axis which is approximately the same as the optical axis of the cat's eye. A bar is drawn in black across a diameter of the disk. Its orientation may therefore be altered to any value.

A microelectrode is inserted into a carefully selected area of the visual cortex of the cat, an electrode fine enough to record the response of a single neuron. The neuron is monitored via the electrode as the disk is rotated. It is found that the firing rate of the neuron will suddenly increase at some particular orientation, going swiftly with angle from the idling rate to a value substantially higher.

Figure 8.14: A cat looking at an oriented bar



We can plausibly argue that the neuron has somehow got a particular sensitivity to edges of the orientation of the bar. Random selection of neurons obtained by simply pushing the microelectrode through, gives a random set of orientations to which the neurons are tuned.

Are the neurons in a ready trained state from the time when the cat is born, or do they acquire a particular orientation at some later date? And what makes them pick one orientation rather than another? These

questions have been addressed, the `cat in the hat-box' experiment described by Blakemore in the paper cited below, shows that when a cat opens its eyes in an environent of vertical stripes, neurons in this area become tuned to vertical edges to a large extent, when the cat is brought up in a horizontally striped environment, it has large numbers of neurons dedicated to responding to horizontal edges, and when it is swapped between boxes in its early life, it acquires neurons which are tuned either to horizontal edges or to vertical edges, but not both.

Next Up Previous Contents

Next: Geometry and Dynamics **Up:** Neural Modelling **Previous:** Neural Modelling *Mike Alder*
9/19/1997

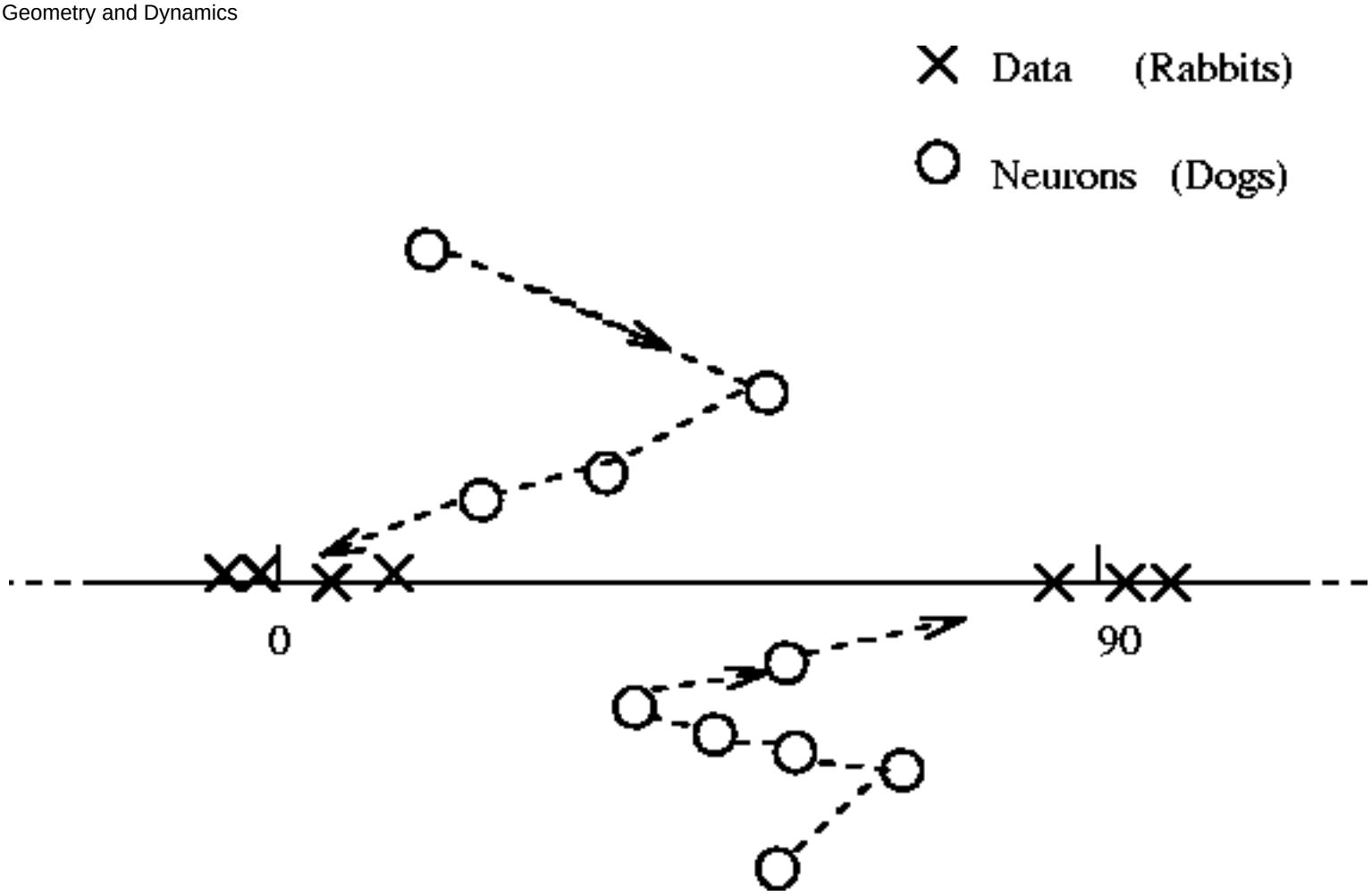
Geometry and Dynamics

Since this happens over fairly short times, it is natural to conjecture that the self-tuning phenomenon observed in the visual cortex of the cat occurs quiet generally and is related to learning and cognition. The situation may be described more colourfully in terms of the neurons being attracted to the data; if we describe the state of the neuron by the input to which it responds most strongly, we can imagine it as a point sitting in the same space as the input data. And when an input datum is presented to the sensorium, the neuron `moves' towards the datum, much like a dog being drawn irresistibly towards a rabbit when the latter sticks his head out of his hole. The neuron doesn't move physically inside the head of the cat, it changes its state so as to move in the state space. In the case of the edge detecting neurons, we draw a line and mark in a point to denote an angle of zero, another to denote an angle of 90 degrees. Then we put a datum in to the system and in the form of a vertical edge by lighting up the corresponding orientation, a point near the 90^o location. The neurons are not initially on the line, since there is no strongly tuned response to any orientation when the kitten first opens its eyes. So we may put the neurons initially at locations off the line representing the input states. The repeated flashing of data points at or near the 90^o location draws in the neurons however, until they are all close to the data presented. Similarly, if data points are presented near the 0^o location on the line, neurons respond by being attracted to the corresponding location. Neurons are attracted to data just as dogs are to rabbits.

There is some scope for enquiry concerning the dynamical laws governing the attraction of neurons to data or dogs to rabbits. If we have two rabbits alternately springing up in a field at different locations A and B, then if a pack of dogs is set loose at random initial positions in the field, what assumptions can we plausibly make concerning the motion? Well, one assumption is that the dogs can see the rabbits but not the other dogs. Presumably neurons can share a field of inputs, but neurons aren't data, and it seems unreasonable that they should be able to respond to each others states. On the other hand, there is a mechanism, lateral inhibition, whereby they can *hear* each other. If the strength of the output of one neuron is interpreted whimsically as the dog barking, then many groups of neurons have output passed between them, so that the output of one is allowed to inhibit the effects of the common input. This allows for more precise tuning, and it also may be interpreted in the geometrical for which I have a decided preference.

In dynamical terms then, the dogs move towards a rabbit when it stands up and presumably stop shortly after it goes down again. They cannot see each other, but they can see the rabbits, bark more frenziedly as they get closer, and can hear each others barking.

Figure 8.15: Neurons being attracted to data



It makes sense for the dogs to move larger distances when they are closer to the rabbits and smaller distances when they are further away, for the reasons discussed in chapter five. If one contemplates a dog which is attracted to each rabbit alternately, with the two rabbits, one at A and one at B, then the dog which simply jumps half way to the closest rabbit, winds up at neither but at the $\frac{1}{3}$ or $\frac{2}{3}$ point along the line joining the rabbits. Moreover a pack of dogs does the same thing. The pack coalesces to a single dog very rapidly, all in the wrong place. Whether discussing dogs in a field or neurons in a state space modelled on a linear space, we are faced with arguments against letting the law of attraction be what is known in informed circles as a contraction mapping. (See any good book on linear analysis.)

It is natural to suppose that the inhibition from surrounding dogs operates by reducing the size of any jump they might make towards a rabbit. We therefore obtain a fairly simple set of functions determining the motion of a dog towards a rabbit or a neuron towards a datum. The options for the qualitative behaviour are not great, as a more sophisticated mathematical analysis will demonstrate. Here I argue only at the intuitive level, which carries reasonable conviction with most.

Other effects which may be expected to affect the dynamics of neurons when attracted to data, are found in biological neurons: habituation dims the response of neurons to repeated data eventually, lability of neurons, their propensity to respond by change of state is believed to be reduced after some time. I suppose in the more formal model that the neuron jump size towards the datum is reduced by a `fatigue' factor which is proportional to the number of data seen (or repeated) close to the present state of the

neuron. This has the desirable effect of locking the neurons on close data and making them immune from the disturbing influence of remote data. A description of the formal system complete with rational function descriptions of the dynamics may be found in the paper by McKenzie and Alder cited in the Bibliography below.

Next Up Previous Contents

Next: Extensions to Higher Order **Up:** Neural Modelling **Previous:** Self-Tuning Neurons *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Layering **Up:** Neural Modelling **Previous:** Geometry and Dynamics

Extensions to Higher Order Statistics

The proposition that a neuron might be in the business of finding the centres of clusters, in dynamical terms, has implications for what we have suggested is the mechanism for UpWriting, as well as general force in the context of unsupervised learning. This is rather different from the model of a neuron as binary classifier or adaptive threshold logic unit. It has far more similarity to the Kohonen network, but that is matter of which I shall not speak, as Gandalf puts it. But can this model be carried further? There are arguments for thinking it can.

A neuron in the visual cortex of a cat is presented with a flood of data concerning the edge orientations painted on the wall of its box: it can do temporal integration and the scene changes continuously over times shorter than the saccadic jumps of the eye. The inputs to a single neuron will almost invariably be correlated, and the neuron may be able to learn this fact. If the modification of synaptic transmitter substance or synaptic geometry, whatever records the effect of the datum on the neuron, is accomplished by a process which admits of interactions between synapses, then the correlations may also be learnt. To put it in geometric terms, the neuron sees not a single datum but a cluster of points in the input space, and the neurone may adapt its state to respond to their distribution. In simple terms, instead of responding to merely the location of the data points, it may be modelling higher order moments than the zeroth and first.

We know that the neuron has a response which is not infinitely sharp, and instead of representing it as a point or dog sitting in the space, trying to sit over a rabbit or cluster of rabbits, we might usefully think of it as conveying shape information. If for some reason we wanted to stop at seond order instead of first, we might visualise it as an ellipse, trying to do a local modelling, and the response to a subset of the data might be modelled by something not too unlike a gaussian distribution specified by the form represented by the ellipse. Thus a family of neurons may be dynamically achieving something rather like a gaussian mixture model of a data set. The complications avoided by stopping at second order make this quite pleasing, but of course there is no particular reason to think that real neurons do anything so simple In general we may however model them as computing moments up to some order less than n, for data in $\mathbb{R}^n$, where n is of order the number of afferent synapses of the neuron. This may be typically around 50,000, according to current estimates.

Next Up Previous Contents

Next: Layering **Up:** Neural Modelling **Previous:** Geometry and Dynamics *Mike Alder*
9/19/1997

Next Up Previous Contents

Next: Summary of Chapter **Up:** Neural Modelling **Previous:** Extensions to Higher Order

Layering

If a family of neurons sharing the same input field from some part of the sensorium may be crudely modelled as a dynamical and sequential form of a gaussian mixture model finding clusters in the data, we have some interesting grounds for speculation about statistical models. I shall avoid this and go straight to the propensity for neurons to form layers, and for the output of one layer to be the input to the succeeding layer. If each layer is accomplishing a model for the local clustering of its input, is a sequence of layers of such families implementing Syntactic Pattern Recognition? It is easy to see that this is not too hard a question to answer in the affirmative. The process of chunking I have described is readily seen to be easily implementable in layers of cluster finding entitities. The details need rather more formalism than I allow myself in this present book, but I claim it can be done. Moreover, the evolutionary process of layer formation can easily be accounted for along the lines discussed earlier, when I pointed out the variation in data that could be accommodated more effectively by having intermediate levels of UpWrite. We can essentially identify the levels of UpWrite with the layers of processing elements.

Next Up Previous Contents

Next: Summary of Chapter **Up:** Neural Modelling **Previous:** Extensions to Higher Order *Mike Alder 9/19/1997*

Next Up Previous Contents

Next: Exercises **Up:** Syntactic Pattern Recognition **Previous:** Layering

Summary of Chapter

The vision of the brain as a machine for, cognitively speaking, finding correlations in its inputs so as to be able to create entities from the sensorial chaos has its attractions. We are, it must be confessed, talking about what has been called *concept formation* here, although the term must fill any mathematician, or even anyone with a decent feel for language, with fury and loathing. The point of `extracting features', to use another abominable contortion of the inarticulate, in this way, is of course to do more than classify, it is to predict, contingently; it is to survive. It would seem however, that the process of extracting and symbolising entities via layers of neurons has to be a fairly basic piece of machinery with which to accomplish survival in this present Universe.

In this chapter I indicated initially the drawbacks of the kind of static recognition I had used at the start of the book, and which are the commonplace methods of the times. I then drew attention to the inspiration of King Sun Fu that there was some similarity between the structure of natural language and that of certain images, and pointed out certain drawbacks in his attempt to provide algorithms for using this insight to understand images. I assured the reader that there was a formal theory couched in Category Theoretic terminology which allowed one to exploit the structures Fu had observed, and I undertook to convey the elements of this theory by practical exampole rather than formal mathematics. The remainder of the chapter consisted of my trying to fulfil that promise, with what success the reader is in a position to pass judgement, and in a final section I indicated that there was a link between the statistical methods of mixture modelling and what neurons might do in the Central Nervous System - in other words, paradoxically, statistical pattern recognition may be closer to what neurons do than neural net pattern recognition. I then suggested that there was a link between the UpWrite processes and the layering of neurons in the Nervous Systems of the higher animals.

The theory of cognitive processes and their implementation which this implies, may be tested empirically by building syntactic pattern recognition systems. There is some prospect of some of the work which has applied these ideas, actually making a few bucks for the present writer in the not too distant future. Very, very few bucks are likely to actually get past the tax man and his emulators. Of course, they would only be spent on Havana cigars, or even less respectable forms of self-indulgence, so perhaps it is no bad thing that the customers are all close to broke, or claim to be.

Next Up Previous Contents

Next: Exercises **Up:** Syntactic Pattern Recognition **Previous:** Layering *Mike Alder*
9/19/1997

Exercises

I think you are entitled to think of your own applications and see if you can make them work using these methods. Some of the papers in the Bibliography may inspire you.

Mike Alder
9/19/1997

Next Up Previous Contents

Next: About this document ... **Up:** Syntactic Pattern Recognition **Previous:** Exercises

Bibliography

1. R. Gonzalez, M. Thomason *Syntactic pattern recognition: an introduction* Addison-Wesley Pub. Co., Advanced Book Program, 1978.
2. K. S. Fu *Syntactic pattern recognition and applications* Prentice-Hall, 1982.
3. T. Pavlidis *Structural pattern recognition* Springer-Verlag, 1977.
4. C. Blakemore *Developmental Factors in the Formation of Feature Extracting Neurons* , pp105-114, in: *Feature Extraction by Neurons and Behaviour* Ed. Gerhard Werner, MIT Press, 1975.
5. P.McKenzie, M.D. Alder, *Initialising the EM Algorithm for use in Gaussian Mixture Modelling.* Pattern Recognition in Practice IV, Vlieland, 1994.
6. M. Alder *Topological Stochastic Grammars* Appeared in ISIT, International Symposium on Information Theory; copy on the CIIPS ftp site: see my home page.
7. G. Lim, M.Alder, C.deSilva *Syntactic Pattern Classification of Moving Objects in a Domestic Environment.* Pattern Recognition in Practive IV, Vlieland, 1994.
8. R. McLaughlinn, M. Alder *Recognising Cubes in Images.* Pattern Recognition in Practice IV, Vlieland, 1994.
9. M. Alder *Inference of Syntax for Point Sets* Pattern Recognition in Practive IV, Vlieland, 1994.

Mike Alder
9/19/1997

...system.

There is a school of thought which believes that this shows beyond all doubt that scientists and engineers are crass. Subscribers to this belief are seldom capable of mastering, say, calculus. Who is the crasser?

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...likelihoods

This use of the term `likelihood' for the value of the gaussian function evaluated at a point is frowned on by statisticians and probabilists, who insist that the likelihood is a function of the parameters of the gaussians (or whatever distributions they wish to use), and not a function of the data. It is obvious that the number obtained is actually a function of two sets of variables, one comprising the data and the other the parameters of the distribution. Statisticians, and some Probabilists, seem to have trouble with this idea and a sentimental attachment to a rather clumsy language. Engineers commonly use the terminology I have employed, and although their jargon is sometimes just as bad, in this case they have mathematics on their side, so I follow their usage.

.

.
.
.

...language

And the extensions of the language, used currently for describing objects studied in theroretical physics, such as vector bundles and tensor fields.

.
.
.
.
.
.
.

...language

At least this is one interpretation of what he thought he was doing. He may have thought that there is some vagueness in the universe.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...hypothesis".

Lagrange is supposed to have piously remarked on hearing this story ``Ah, but it is such a beautiful hypothesis, it explains so much." Actually, it explains everything. That's what's wrong with it.

.
.
.

...them

The best bet at the time of writing, is to get yourself a nice Pentium machine and install LINUX, thus getting the best of both worlds.This text is being written on such a machine, and will be compiled by LaTeX. The LaTeX is explained in the Scientific Writing course, and the installation of LINUX should be done by consultation with your tutor.

.
.
.
.
.
.
.
.

...levels

And for the lucky among us, colour.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...book.

I promised that this book would be user friendly and am trying to keep my word. Maybe my next book will be academically respectable and properly inscrutable.

.

...us.

It is rumored that Probabilists are good people with whom to play poker, because they focus on the probabilities and not the personalities. I am unable to confirm this from personal experience as none of the Probabilists I know will play poker with me. I can't imagine why.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...board,

It is usual to model dart throwing as selecting a point at random inside a disk.

This might shock genuine darts players. Of course, real dartboards are made of atoms, and space might come in little lumps for all we know, so this model is only one of several.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.

...the

The main objection to starting with a *pdf* is that not all measures can be represented by *pdf*s. We can assign a uniform measure to the real line, and also the measure which concentrates every possibility at one point. These are perfectly respectable measures and might well be of use in probability theory, but they cannot be described by *pdf*s. One way out of this is to allow ourselves to use *generalised functions* to represent measures, where a generalised function is a suitable sequence of ordinary functions. For instance, we can take a gaussian on the origin and make it take on progressively larger and larger values for the standard deviation, to get a sequence of functions defining the uniform measure on the real line. Or we could make σ tend to zero so as to get the measure concentrated at the origin. To be more precise, what I should have said is that it would be simpler to start with a measure on the sample space of possible outcomes. This is often done. I shall assume from now on that we are free to use generalised functions to represent measures, or that we can restrict ourselves to the case where the measure can be represented by a (usually continuous) function.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.

...pdf.

There is a school of thought which regards this as a very wicked choice of terminology. This is because probabilists have persuaded themselves that the right way to use the term `likelihood' is to mentally drive a nail through the data and then regarding the numbers obtained as a function of the parameters used for specifying a *pdf*. *This* is the likelihood. They are, however, making rather a pointless distinction, since they are talking about a function of two variables, data, and parameters of a family of *pdf*s. What I describe as pointless, they would describe as subtle. My objection to this particular subtlety is that it seems to be concerned with the semantics rather than the syntax: they use the term so as to tell you what they plan to do with the number, not how they compute it. Conversely, when offered my definition above, probabilists have assured me disdainfully that there aren't any parameters, so my definition makes no sense. But there is a perfectly good parameter space, it just happens to consist of a single point. I have discussed this matter with distinguished probabilists: usually, each of us winds up thinking the other rather crass. I may return to this point later if my conscience gets to me.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.

...randomness

This leads to the following argument: Of all the sequences that could occur in a game of poker, some are random and others are not. Without getting into details, all hearts first, in order, then clubs, then diamonds, then spades, is one of the non-random sequences. Let us therefore remove these non-random sequences from consideration. Our cards have been shuffled, and if the sequence turned out to be non-random, a complaint that the shuffler had failed to do his job properly would carry. Hence we may assume that our sequence of cards is one of the random ones.

The arguments advanced in the beginning of this chapter on the probability of filling a flush or a straight at poker are now invalid. They depended on considering all sequences of cards as equally likely, but we have deliberately excluded the non-random ones. How much difference this would make in practice is a little hard to say *absent* a definition of random.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.

...stated,

But see Thom's *Structural Stability and Morphogenesis*, and Abraham's *Foundations of Mechanics*.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...Penttila,

In the common room of the Mathematics Department at the University of Western Australia. The coffee is awful, but the conversation sometimes makes up for it.

.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...Model.

This terminology is more used by engineers than probabilists, for whom the term `model' is more often used to denote what I have called a family of models. Thus a probabilist would commonly say that what I have called a family of models indexed by the numbers between 0 and 1 is just the binomial model. Since it is clearly a good idea to have a distinction between the space of things and the individual things in the space, I shall follow what seems to be a *de facto* tradition among engineers and such roughnecks, that is less confusing than the tradition followed by statisticians and probabilists. Beware of this point when reading the works of the latter.

.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...model

Known to followers of E.T Jaynes as the `SureThing' model.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.

...past.

What `roughly similar' means is, of course, intimately tied up with transfer of learning and the crucial business of what precisely constitutes a replication of an experiment? These issues are slithered over in the foundations of probabilistic statistics because they are too hard.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...English.

It isn't that easy, but this is another subject.

.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...Inequality.

Proofs done using water are frowned upon in some quarters, usually by people to whom proofs are part of life's mystery, and who want to keep life as mysterious as possible. You may have read some of their books, or tried to.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...Platonists.

Of whom there are a fair number around. No doubt a cure will be found in due course.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.

...sort)

The Intermediate Value Theorem tells you that if you have a continuous function defined on an interval which takes values which are positive at one location and negative at another location, then it must be defined at some other location where it takes the value zero. This codifies an intuitive picture of the real line as resembling a ruler made out of chewing gum and a continuous function as stretching it but not tearing it, and it means that when we want to find the zero of a function by some iterative process, we have usually got grounds for thinking it is there to be found. The price paid for this simplicity and adherence to naive intuitions includes the uncountability of the reals and non-constructive existence proofs which give confidence only to the more naive Applied Mathematicians.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...for

Well, it's what *I* want models for. If you have some alternative use for them, good luck to you.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...rapidly

Well, fairly rapidly. Doing this with axiomatic precision requires careful avoidance of pathologies which trouble the sleep of Pure Mathematicians but which most people wouldn't think of. I shan't ask you to think of them either.

.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
...task.

Robots *are* programmed to do similar sorts of things, e.g. shoot down enemy aircraft, and they do it tolerably well.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.

...model.

There is altogether too much of people feeling things in their bones in this business.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...cheese?

Do *not* try to use it to prove Kraft's inequality.

.
.

...chance

Actually, probability 1.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...meanest

If your intellect isn't very mean, and if these things are not immediately obvious, you need some remedial linear algebra.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.

...small.

Of course, I can compress this information better if I know anything about it, which is the point of modelling the data. Likewise I can compress the parameters better if I know anything about how *they* are distributed, which gives us the Bayesian prior. And smaller fleas...

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...utility.

It is true that no computer can add any two numbers. The point is that there are always numbers too big to be represented in the computer, no matter how big the computer is. Since we can make computers which can add numbers in the range we usually care about, this is not quite so devastating as it sounds to the innocent and uncritical.

.
.

...military

That is, he hadn't killed anyone yet.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...down,

Or a particularly half baked one either, alas. Astrology has been going much longer than Science, which may yet turn out to be a passing fad.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.

...smooth

`Smooth' is Mathematician's talk for differentiable as many times as looks like being a good idea

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...class.

S^0 is the zero dimensional sphere for those who don't know; the n-dimensional sphere is the equator of the n+1-dimensional sphere and the 1-dimensional sphere is a circle. The equator of a circle consists, of course, of the two points $\{\pm 1\}$. Now you know.

.
.
.
.

.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...hyperplane.

This is intuitively hard to doubt and if you wish to doubt it, do so, but be careful, you may turn into a Pure Mathematician.

The price of freedom to doubt intuitively appealing assertions like this is that you get asked to prove them. Go ahead, make my day.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.

...time.

This is a legitimate trade off, of course. The extent to which people opt for random searches through huge state spaces, rather than think about what they are doing, tells you how high a price they need to be paid in order to think. For some people, there isn't that much money.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...support

compact support means that the function is zero except in some bounded region of $\mathbb{R}^n$.

.

...easy

Well, fairly easy. See Simmons in the references for a perspicuous proof.

...Rock

For those readers unfortunate enough to live somewhere other than Western Australia, it should be explained that Wave Rock is a tourist attraction. It is a rock that looks quite a lot like a wave on the point of breaking.

.
.
.
.
.

...Guinea.

Or, for that matter, the wilder parts of New York.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...Recognisers.

Some of us are humbler than others, of course. And some of us have a self-esteem that would survive impact by a medium size asteroid.

.
.
.
.

.
.
.

...

That's all from Chris. You're back with me again from now on.

.

...alphabet

If this looks like a concession to the Artificial Intelligentsia, and a descent into rather crude types of representational machinery, this is an error, and I shall put you straight in later chapters, but it has to be done, if only because continuous trajectories are reduced to discrete trajectories a lot of the time. There are other and better reasons, as the persistent reader will discover.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...frequency

The same effect can often be obtained by using a cheap microphone, but this approach is thought unprofessional by most engineers. They prefer to get an expensive microphone and then put a filter in to make it behave like a cheap one.

.
.
.
.

...range

By comparison, the human ear has a cochlea, a rubbery tube filled with fluid, and a membrane down the middle of it, which vibrates when sound strikes the ear, and which because of varying properties of the membrane, responds in different places to different frequencies. The vibration of the basilar membrane is monitored by around 30,000 hair cells. It is thought that this performs something like a fourier transform on the speech signal, but it is known that much more happens than this. The difference between 16 and 30,000 is a little disturbing.

...function

If you can't tell the difference, don't cross any roads until you have thought about it.

.
.
.
.
.

...way

So much so that one of my more ebullient colleagues has been found threatening to vector quantise people who argue with her.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...algebra:

Watch out for the next footnote.

.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...procedure

Described in English rather than algebra. Algebra has the merit of precision and the drawback that it can take a long time to work out what the underlying ideas are, sometimes rather more than a lifetime. It is generally rather easier to turn an idea from English into algebra or a suitable programming language than to go the other way round, but you may memorise algebra and get the illusion that you understand it when you have merely learnt it. Engineers sometimes use algebra as Lawyers and Medical men use Latin: to impress the peasantry and justify their salaries. Also, it conceals the fact that they can't spell and have awful handwriting.

.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...hope.

I suppose that a methodology which allows morons to publish research papers is a fine and egalitarian thing, but it does clog up the journals somewhat. Again, I incline to blame the Mathematicians for a disgracefully inadequate job of teaching Mathematics in Universities particularly to Engineers and Physicists. Of course, the Mathematicians blame the Engineers and Physicists for demanding technical skills without any idea of why the methods work. This leads to Engineering journals keeping the standards high by ensuring that there is a significant mathematical content to each paper. They decide this by counting the number of integral signs, partial derivatives and half the summation signs and divide by the number of pages. This leads to the most banal of ideas, which could easily be explained in English in half the space with twice the clarity, being expressed in algebra. The other part of the problem is an insane ambition which leads folk of limited intellect to imagine that they can do Science and Mathematics. Let them do the so called Social Sciences. That's what they were invented for. Real Science and Engineering and Mathematics are difficult. Ah, I feel lots better now, thank you. You might ask why diatribes like this are found in a simple, academic book. Well, if nobody bitches about incompetence, dogmatism and stupidity, we shall get a lot more of it, and there's already plenty. It's a dirty job, but someone has to do it; it clearly takes a highly developed contempt for the whole concept of political correctness, making the author rather well qualified. Actually, I'm joking. Academic life is really just one big happy family.

.
.
.
.
.
.
.

.
.
.

.
.
.
.
.
.
.
.
.
.

...trying

There are two activities traditionally practised by grammarians. The first is to regulate how other people ought to talk and write. Grammars promulgated for this reason are called *normative*. The second is to try to describe what is actually said or written by some group of people. Such grammars are called *descriptive*. We are only concerned with the second sort.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...extensively.

It is important to understand that we don't judge a model by its plausibility in science, we judge it by whether it works or not. Thus we replace woolly arguments between proponents and detractors by careful investigation of the world. This saves a lot of time, and sometimes blood.

.
.
.

.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.

...limited.

This is the *good* news.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...equations.

The power of Linear Algebra is shown here: No less than four blokes got their names attached to a geometrically trivial observation just because the dimension was higher than three.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...oafs

Well, the ones who can put up with me tend to be. No doubt it's a biased sample. Of course, *they* say they are merely healthy and have good digestions and clear consciences as a result of doing honest work.

.
.
.
.
.
.
.
.

.
.
.

.
.
.
.
.
.
.
.
.
.

...see.

You were warned about function fitting. Don't say you weren't.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...`seminal'

Others have been even ruder.

.
.
.

.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...streets.

This is important. The damage those guys could do if let loose in business or politics doesn't bear thinking about.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.

...bibliography.

It may be remarked that many books on languages and automata are written by mathematical illiterates for mathematical illiterates, which makes them hard to read, and Eilenberg is a notable exception. His book is hard to read for quite different reasons: he assumes you are intelligent.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...on.

After doing it my way, I discovered that Eilenberg had got there before me, and he explains how somebody else had got there before him. Such is Life.

.
.

.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...indeed.

If you are used to reading chinese you may feel differently about this, but then I expect you are having a lot of trouble reading the present work.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.

...long.

The reason for allowing infinitely long sequences is not that anybody expects to meet one and be able to say with satisfaction, `wow, that was infinitely long', it is more of a reluctance to specify the longest case one expects to get. In other words, it is done to simplify things; the infinitude of the natural numbers is a case in point. There may be some grounds for reasonable doubt as to whether this in fact works: sometimes one merely defers the difficulties to a later stage.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...end.

Grammarians used to believe that high levels of inflection showed a highly evolved language, probably because Latin was regarded as morally superior to English, German or French. Then it was discovered that Chinese used to be inflected a few thousand years ago but the chinese very

sensibly gave it up.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...level.

Past a certain point it is called `plagiarism'.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...is.

As Herod said to the three wise men.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.

...methods

Or whatever is the pet method favoured by the audience.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...world.

To a geometer, it looks as though the algebraists have gotten into the cookie-jar when everyone knows they are there to tidy things up and nit-pick about details, not go around being creative. Or, even worse than algebraists, gasp, *logicians*. When logicians go around being creative, universes totter and crumble.

.
.

.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...world.

As when one's progeny want one to switch off the Bach so they can concentrate on the Heavy Metal, or *vice versa*.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.

...group.

A *Topological Group* is a collection of invertible transformations such that the composite (do one then another) of any two is a third, and also having the property that the multiplication operation and the inversion operation are continuous. A *Lie Group* (pronounced `Lee') is a topological group where the operations are also differentiable and the set forms a manifold, which is a higher dimensional generalisation of a curve or surface. This definition is sloppy and intended to convey a vague and intuitive idea which is adequate for present purposes.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...group

The term is from differential topology, and may be found in any of the standard texts. It is not necessary to expand upon its precise meaning here, and an intuitive sense may be extracted from

the context.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...book.

But watch out for the sequel.

.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

...cognition.

Blakemore conjectured it in the paper cited, but it probably struck a lot of people that this was a variant of a Hebbian learning rule.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.

...preference.

There is a belief in some quarters that God is a mathematician, or why did He make Physics so mathematical? And in particular, He must be a geometer, because most of Physics uses geometry. This is like wanting the Sun to shine at night when we need it and not in the daytime when it's light anyway. The fact is that the most powerful languages allow you to say more interesting and important things. *That* is why I use geometry.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

Mike Alder
9/19/1997

Next Up Previous Contents

Up: An Introduction to Pattern **Previous:** Bibliography

About this document ...

An Introduction to Pattern Recognition:
Statistical, Neural Net and Syntactic methods of getting robots to see and hear.

This document was generated using the **LaTeX**2HTML translator Version 97.1 (release) (July 13th, 1997)

Copyright © 1993, 1994, 1995, 1996, 1997, Nikos Drakos, Computer Based Learning Unit, University of Leeds.

The command line arguments were:
latex2html `PatRec.tex`.

The translation was initiated by Mike Alder on 9/19/1997

Mike Alder
9/19/1997